第1章 传 统 密 码

为了经济或军事的需要，古代就出现了各种各样的加密方法。经过人类多年的研究和改进，已经取得了大量的成果。现代密码学是在借鉴与发展传统密码学的基础上诞生的。在学习现代密码学之前，了解传统密码学的基本概念和成功经验是十分必要的。本章将介绍最典型的几种传统密码算法及其破译方法。

1.1 基 本 概 念

1.1.1 密码与密码学

密码是为解决信息安全而进行的编码。安全指通信系统抗御外来攻击的能力。外来攻击主要有两大类：一类是以截获或窃听通信内容为目标的被动攻击，攻击者截获他人信息、窃取密码、打探隐私、偷盗机密、危害民众；另一类是以篡改或伪造信息为手段的主动攻击，攻击者冒充合法发信人，发布信息，安置黑客，散布病毒，甚至破坏通信系统。针对被动攻击，密码可以使所传输信息具有保密功能，窃听者即使截获了一些信息，也会因不懂密文而不知所云。针对主动进攻，密码应具备“认证”功能，对发信人身份、消息来源以及消息完整性等加以认证，使非法发信人不得进入系统，使虚假消息能被识别，使篡改行为得以被发现。保密和认证是密码的两大基本功能。

为了军事、政治、司法等方面的需求，有时也需要破译对方的密码或者赚取对方的认证，由此便出现了与“密码编码学”原理相同但目的相反的另一分支，叫做“密码分析学”。这种保密与破译的斗争如同矛与盾的关系一样，魔高一尺，道高一丈，相依相存，相克相长，促进了二者共同的发展。近年来曾多次听到某种保密系统悬赏破译，某个防火墙产品欢迎投诉，其目的就是通过不断发现问题与解决问题，增强自己的抗攻击性能。为了设计出更加安全可靠的密码系统，设计者不仅要研究出更新更强的密码技术，还要研究对手有哪些可能的攻击手段。设计以分解大数复杂性为基础的 RSA 密码体制时，必定要讨论分解大数的技巧；设计以离散对数复杂性为基础的密码体制时，难免要讨论离散对数的计算方法，因此，聪明的密码设计者同时也应该是高明的密码分析者，二者统一于同一个目标。“密码学”则是“密码编码学”与“密码分析学”的总称。

1.1.2 传统密码学与现代密码学

密码技术的历史源远流长。很久以前，人们为了保存或传递经济、军事、外交上的重

要信息，就已经开始使用密码或类似的保密技术，发明了多种多样的加密和解密方法，其手段由手工加密发展到机械加密，直到近代的电子加密。密码技术的发展历史上，记载着不少精辟的成就，书写了丰富的经验，密码技术的成果也曾在二战中发挥了巨大作用。然而直到 20 世纪 40 年代以前，密码技术却一直是纯经验性的学问。1949 年，Shannon 发表了“保密系统的通信理论[1]”，用信息论的观点对密源、密钥、密文等概念进行了数学描述，对保密系统进行了定量化的分析，才使保密学建立在科学的理论基础之上。

我们把 20 世纪 70 年代以前的密码研究与应用称为传统密码学。那时，互联网、智能卡以及很多现代交互方式都还没有出现，传统密码学以研究秘密通信为目的，它关注的是怎样使所传输的信息不被第三者所窃取，发信人身份的认证问题并未引起足够的重视。

随着计算机互联网的出现与普及，一方面通信网络极大地方便了人们对信息的交换和共享，另一方面也同时给攻击者提供了更多的机会。如何更好地解决信息安全问题，已成为刻不容缓的任务。社会需求的推动与科研成果的积累，终于迎来了 20 世纪 70 年代以公开密钥体制为标志的密码学大发展阶段。古老的密码学重新焕发出蓬勃的生机，成为新的热门学科。密码学作为信息安全的卫士，已走出密码学家神秘的殿堂，成为广大民众和商界关注的学科。

一般认为现代密码学诞生于 20 世纪 70 年代，标志性的大事有两件[2]：一是 1977 年美国国家标准局正式公布实施了美国数据加密标准(DES)，并批准用于非机要部门和商业用途，传统密码学的神秘面纱被揭开；二是 1976 年 11 月，美国斯坦福大学电气工程系研究生 W. Diffie 和副教授 Helman 在 IEEE 上发表了题为“密码学新方向”的学术论文，公钥密码研究的序幕就此拉开。从此密码学以一种全新的理念和姿态迅速崛起，理论研究成果频出的同时，实际应用也大踏步地走进了人们的生活。

传统密码学只重视保密功能，而现代密码学除了重视保密功能之外，还对认证功能提出了多方面的要求，如发信人身份认证、信息完整性认证、不可抵赖性认证等，对于抵御主动攻击发挥了巨大的作用。数字签名是一种常见的认证手段。现代密码学还提出了公开算法的思想，开创了公开密钥体制的新思路，从而在安全观念与设计理念上明显地区别于传统密码学，独树一帜，成为当代信息安全的重要技术支柱。由于现代密码学的需要，过去某些纯数学理论的领域，如经典数论、椭圆曲线等，一下子找到了用武之地，变得火爆起来。

1.1.3 对称密钥与非对称密钥

就系统使用的密钥来分类，有两大类密钥体制。一类是对称密钥体制，加密与解密的密钥相同，掌握加密密钥的人，必然也能解密，这种密码体制也叫做单密钥体制。另一类是非对称密钥体制，加密密钥与解密密钥是不相同的，因而可以将其中一把密钥公开，只需秘密保管另一把，这种密码体制也叫做公开密钥体制或双钥体制。传统密码系统全都属于对称密钥体制，而现代密码系统中两类密钥体制都存在。比如 DES 属于对称密钥体制，RSA 则属于非对称密钥体制。

1.1.4 分组密码与序列密码

就系统加密与解密方式来分类，有分组密码与序列密码两类。分组密码是把欲加密的

文本分成一些较短的段落，每次使用相同的密钥与算法处理一段，然后将处理后的各个小段连接起来。解密过程也是按原来的分段一段段解译，然后再连接起来。序列密码方式则不进行分段，直接把整个文章顺序送入加密器，密钥也应当源源不断地输入加密器，加密器通常只是完成某种很简单的算法，比如模二加法。这种看似简单的一字一密加密方式已被 Shannon 从理论上证明是无法破译的，当然它要求的密钥是无限长的随机序列。

1.1.5 密码学基本术语[3]

保密通信过程的流程图如图 1.1 所示。

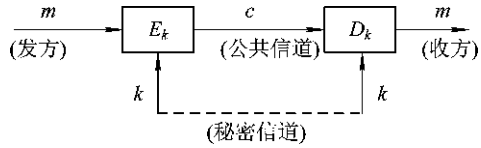

图 1.1 保密通信的流程示意

保密通信的一些基本术语对于传统密码学与现代密码学都同样有用。如：

· 明文(plaintext)：发送方(sender)未经过加密处理的信息，其内容是容易理解的。

· 密文(ciphertext)：发送方经过加密处理后的信息，文字被改变，其内容是难以理解的。

· 密钥(key)：发送方加密处理时所用的秘密信息。在传统密码学中，解密也使用同样的密钥。

· 加密(encryption，encipher)：发送方把明文加工成密文的变换，即

$$c = E_k(m) \tag{1-1}$$

· 解密(decryption，decipher)：接收方(receiver)把密文解译成明文的变换，即

$$m = D_k(c) \tag{1-2}$$

为了顺利进行上述保密通信过程，通信之前还必须完成以下准备工作：

(1) 协议(protocol)：约定通信双方的通信步骤和各个技术细节。

(2) 密钥交换(key exchange)：通信双方必须设法取得所约定的密钥(指对称密钥)。用公共信道传输密钥是不安全的，除非利用某种专门用于传送密钥的秘密信道来传输，然而其代价可能是昂贵的，或是很不方便的。可见，密钥交换问题是传统密码体制的一大难题。

1.2 传统密码举例[2]

1.2.1 逆序密码

设：明文为

m=Thousands of years ago, cryptography had been used to keep secrets of military operations or treasure.

加密算法 E_k 为将原文忽略空格并不计字母大小写，每 5 字符一组，各组取逆序，连接后得到密文：

c=suohtosdnaraeyfcogasotpyrhpargbdahysuneekotdeespeesterclimfoyratiareposno-itertroerusa

这里，$k=5$ 是密钥。

1.2.2 棋盘密码

将 26 个字母写入 5×5 方阵中，i 和 j 占同一格，使每个字符都由一对行、列脚标表示之，如表 1.1 所示。例如，明文为

m=There are all kinds of encryption schemes in classic cryptography.

则将原文忽略空格，用表 1.1 的行、列脚标编码，可写出如表 1.2 所示的密文。

表 1.1 棋盘密码的字母排列

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	i, j	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

表 1.2 棋盘密码对明文的加密结果

44	23	15	42	15	11	42	15	11	31
31	25	14	33	14	43	34	21	15	33
13	42	54	35	44	24	34	33	43	13
23	15	32	15	43	24	33	13	31	11
43	43	24	13	13	42	54	35	44	34
44	34	22	42	11	35	23	54		

1.2.3 凯撒密码

将 26 个英文字母按字母表序号 0～25 编号（有时也可以按 1～26 编号），如表 1.3 所示。加密编码的方法是把明文的每个字母用向后循环移动 k 位后的字符代替：

$$c = E_k(m) = m + k \bmod 26 \tag{1-3}$$

移位距离 $k(0<k<26)$ 是密钥。

表 1.3 英文字母按字母表序号 0～25 编号

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

例如明文是：

m=Julius Caesar（朱丽叶斯·凯撒）

则不计空格与大小写，明文各字符的序号为

9 20 11 8 20 18 2 0 4 18 0 17

假设平移 $k=11$，则变为

20 31 22 19 31 29 13 11 15 29 11 28

循环位移意味着：

$31=5 \bmod 26$，$29=3 \bmod 26$，$28=2 \bmod 26$

于是，得到平移后的序号：

20 5 22 19 5 3 13 11 15 3 11 2

再按字母表写出，则密文为

c=UFWTFDNLPDLC

更一般的方法是不仅平移而且作“仿射变换”：

$$c = k_1 m + k_2 \bmod 26 \tag{1-4}$$

式中，k_1 和 k_2 是两个密钥，k_1 要求与 26 互素。

例如，m=information 对应的字母序号是：

8 13 5 14 17 12 0 19 8 14 13

若选 $k_1=7$，$k_2=10$，则变换后的数字是：

14 23 19 4 25 16 10 13 14 4 23

对应的密文是

c=OXTEZQKNOEX

1.2.4 单表置换密码

实际上，除了移位，还可以置换，甚至任意调换，26 个字母的各种排列共 26！种，每指定一种调换方式，都可以写出一张与原字母表对应的置换对照表，称为单表置换。在置换表中引入密钥的方法有许多种。例如可按下述方式编制置换对照表：设密钥为 University of Science and Technology，略去重复字符，得到 UNIVERSTYOFCADHLG，排在置换对照表的最前面，后面顺序接上字母表中尚未出现过的字符，就构成了下面的列置换表：

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
U	N	I	V	E	R	S	T	Y	O	F	C	A	D	H	L	G	B	J	K	M	P	Q	W	X	Z

1.2.5 多表置换——维吉利亚(Vigenere)密码

单表置换虽然改变了明文的模样，但密文中同一个字符总来自明文的同一字母，这样就很容易从概率分析上破译(找到概率最大的字母，就可能是 e 变来的)。为此引入多表变换，原文中的同一字母出现在不同位置时，会变换成密文的不同字符。

设密钥 k 是长度为 n 的字符串 $(k_1k_2\cdots k_n)$，这里 k_j 是密码的第 j 个字符在字母表的序号。明文 m 也被分成许多长为 n 的小段，第 i 段 $M_i=m_1m_2\cdots m_n$；那么该段明文对应的密文 $C_i=c_1c_2\cdots c_n$。则有

$$c_j = (m_j + k_j) \bmod 26 \qquad j = 1, 2, \cdots, n \tag{1-5}$$

例如：密钥 k=shift，其字母序号为 18，7，8，5，19；密钥长度 $n=5$。将明文 m=encode algorithm 忽略空格，按 $n=5$ 分段后，其字母序号是：

4，13，2，14，3； 4，0，11，6，14； 17，8，19，7，12；

则第 1 段 5 个字符分别移位 18，7，8，5，19 后序号变为 22，20，10，19，22，即 WUKTW；第 2 段 5 个字符再分别移位 18，7，8，5，19 后序号变为 22，7，19，11，7，即 WHTLH；第 3 段 5 个字符又分别移位 18，7，8，5，19 后序号变为 9，15，1，12，5，即 JPBMF。最终的密文是 c=WUKTWWHTLHJPBMF。

为了容易地查到不同密钥字符的变换结果，可以列出一张“维吉利亚”方阵表，第一行 26 个字母按序排列。第二行是第一行左移一位的结果，B 开头，字母顺序不变，直到 Z，最后是 A 作结尾。第三行是循环左移二位 ……直到第 26 行以 Z 开头，后接 AB……XY。当

密钥字符为 a 时，就查第一行；当密钥字符为 b 时，就查第二行…… 当密钥字符为 z 时，就查最后一行。表中的列用来查询每个明文字母应当对应什么密文字母。这种方式称为多表置换。

后来，比欧福特(Beaufort)将“维吉利亚”方阵表的各行按逆序排列得到了“比欧福特方阵”，使破译更加困难。

1.2.6 维尔南(Vernam)加密算法

当明文、密文、密钥均为二元代码(0，1)时，维吉利亚密码就成了维尔南密码。维尔南密钥在计算机代码中得到广泛应用，其计算公式是：

$$c_i = (m_i + k_i) \bmod 2 \qquad i = 1, 2, 3, \cdots \tag{1-6}$$

式中，m_i、k_i、c_i 都是 0 或 1。

密钥一般是取一个周期很长的伪随机二元码序列。加密后，密钥的随机性掩盖了明文的可读性，使密文变的不可理解。为了得到较长的密钥，可以找两个不太长的密钥，各自循环着使其中一个对另一个加密，当两密钥长度互素时，可得到长度为二者之积的密钥。

1.2.7 普莱费厄(Playfair)加密算法

普莱费厄加密算法把引入密钥词组的单表密码与棋盘密码结合。例如密钥是 five stars，按单表方式排序后写成 5×5 方阵：

F	I，J	V	E	S
T	A	R	B	C
D	G	H	K	L
M	N	O	P	Q
U	W	X	Y	Z

若明文为

M ＝Play fair cipher was actually invented by wheat stone

则先将明文两两分组：

pl，ay，fa，Ir，ci，ph，wa，sa，ct，ua，lq，ly，in，ve，nt，ed，by，wh，ea，ts，to，ne

分组时注意：

(1) 若分组出现两字母相同时，则在两字母之间插入一个 q。

(2) 若总字符个数为单数，则在最后那个不配对的字母后添一个 q。

加密方法如下：

(1) 若分组 $m_1 m_2$ 在方阵同行时，密文 $c_1 c_2$ 分别是 m_1 和 m_2 右面的字母，其中第一列可视为第五列的右面。

(2) 若分组 $m_1 m_2$ 处在方阵同一列时，密文 c_1 和 c_2 分别是 m_1 和 m_2 下面的字母，第一行可视为第五行的下面。

(3) 若 $m_1 m_2$ 不同行也不同列时，c_1 和 c_2 是 $m_1 m_2$ 为对角的矩阵块的另两个角，并且 c_1 与 m_1 同行，c_2 与 m_2 同行。

按此加密方法，上例中分组被加密为

QK，BW，IT，VA，AS，OK，IG，IC，TA，WT，QZ，KZ，
AW，ES，MA，FK，KE，XG，IB，CF，RM，PI

1.2.8 希尔(Hill)加密算法

将明文中长为 L 的字符分组 m 通过线性变换 k，变为密文中长为 L 的字符分组 c。字符分组以列矢量表示：

$$\boldsymbol{c} = \boldsymbol{k} \cdot \boldsymbol{m} \bmod 26 \tag{1-7}$$

有时，字符分组以行矢量表示，此时应采用的变换式为

$$\boldsymbol{c} = \boldsymbol{m} \cdot \boldsymbol{k} \bmod 26 \tag{1-8}$$

例如，m=Hill，四个字母序号为

$$7，8，11，11$$

若

$$\boldsymbol{k}=\begin{pmatrix} 8 & 6 & 9 & 5 \\ 6 & 9 & 5 & 10 \\ 5 & 8 & 4 & 9 \\ 10 & 6 & 11 & 4 \end{pmatrix}$$

则

$$\boldsymbol{c}=\boldsymbol{k}\cdot\boldsymbol{m}=\begin{pmatrix} 8 & 6 & 9 & 5 \\ 6 & 9 & 5 & 10 \\ 5 & 8 & 4 & 9 \\ 10 & 6 & 11 & 4 \end{pmatrix}\cdot\begin{pmatrix} 7 \\ 8 \\ 11 \\ 11 \end{pmatrix}=\begin{pmatrix} 24 \\ 19 \\ 8 \\ 23 \end{pmatrix} \bmod 26$$

即

$$\boldsymbol{c}=\mathrm{YTIX}$$

解密可由反变换：

$$\boldsymbol{m} = \boldsymbol{k}^{-1}\boldsymbol{c} \bmod 26 \tag{1-9}$$

来计算，k^{-1}是 k 的逆矩阵，且作模 26 处理。掌握密钥的用户不难求出 k^{-1}。本例中：

$$\boldsymbol{k}^{-1}=\begin{pmatrix} 23 & 20 & 5 & 1 \\ 2 & 11 & 18 & 1 \\ 2 & 20 & 6 & 25 \\ 25 & 2 & 22 & 25 \end{pmatrix}$$

1.3 密码分析举例

以破译密码为目的的学问叫密码分析学。越来越多的事实告诉我们，密码分析是密码学不可分割的组成部分。每研制出一种新密码的同时，就应当讨论它的抗攻击性能；每当某种密码被破译的同时，也就启示密码学家发现原先密码系统的漏洞和改进方法。

根据掌握数据的多少，密码分析可以分为以下几种：

(1) 唯密文攻击：仅掌握若干被同一密钥和同一加密算法得到的密文，想导出明文。

(2) 已知明文攻击：不仅掌握若干密文，还知道相应的明文，想从中找到密钥，从而对任何其他同类加密的密文都能破解。

(3) 选择明文攻击：不仅掌握密文，还有多个可供选择的明文。显然攻击力度更强了，或破译更容易了。

下面举几个破译传统密码的例子。

1.3.1 对单表置换密码的分析

单表密码系统的漏洞在于明文中相同的字符一定对应密文中同样的字符，明文中出现概率最高的字符，必然对应于密文中出现最多的字符，因此利用自然语言文字的概率统计性质容易找到明、密文字符的对应关系。下面以两个实例来展示分析过程。

对大量英语文章的统计发现，各个字母出现的概率是不相同的。对单个字母的统计结果列于表 1.4 中(概率值为 $x\%$)。

表 1.4 英文文章中各个字母出现的概率 (%)

字母	a	b	c	d	e	f	g	h	i	j	k	l	m
概率	8.56	1.39	2.79	3.78	13.04	2.89	1.99	5.28	6.27	0.13	0.42	3.39	2.49
字母	n	o	p	q	r	s	t	u	v	w	x	y	z
概率	7.07	7.97	1.99	0.12	6.77	6.07	10.5	2.49	0.92	1.49	0.17	1.99	0.08

【例 1】 已知仿射密码的密文为[4]

FMXVEDRAPHFERBNDKRXRSREFMORUDSDKDVSHVUFEDKAPRKDLYEVLRHHRH

共 57 字。对应的明文是什么？

解：先统计各字符(特别是高频字符)出现的频度：

R(8)，D(7)，E、H、K(各 5)，F、S、V(各 4)

(1) 根据出现频率大小，不妨设 R→e，D→t，R 序号为 17，D 序号为 3，e 为 4，t 为 19。将之代入加密的仿射方程：

$$y = e_k(x) = ax + b \bmod 26 \tag{1-10}$$

得

$$\begin{cases} 4a + b \bmod 26 = 17 \\ 19a + b \bmod 26 = 3 \end{cases}$$

解得

$$\begin{cases} a = 6 \\ b = 19 \end{cases}$$

由于 $\gcd(a, 26)=2$，表明此密码不正确(正确的 a 应当与 26 互素)。

(2) 重新假设 R→e，E→t，E 的序号为 4，从而得

$$\begin{cases} 4a + b \bmod 26 = 17 \\ 19a + b \bmod 26 = 4 \end{cases}$$

解得

$$\begin{cases} a = 13 \\ b = 9 \end{cases}$$

由于 $\gcd(a, 26)=13$，因此还是不正确。

(3) 再次假设 R→e，K→t，K 序号为 10，从而得

$$\begin{cases} 4a + b \bmod 26 = 17 \\ 19a + b \bmod 26 = 10 \end{cases}$$

解得

$$\begin{cases} a = 3 \\ b = 5 \end{cases}$$

由于 $\gcd(a, 26)=1$，因此有可能是正确的结果。

(4) **由加密算法**：

$$y = 3x + 5 \bmod 26$$

得到解密算法

$$x = \frac{y-5}{3} \bmod 26 = 9y - 19 \bmod 26$$

式中：$3^{-1} = 9 \bmod 26$(3 **的模逆元是** 9)。

(5) **用这个结果解译密文，看能否得到有意义的明文。结果是**：

algorithms are quite general definitions of arithmetic processes

得到了有意义的明文，表明破译正确。

【例 2】 已知单表加密的 280 **字密文为**[2]

GJXXN GGOTZ NUCOT WMOHY JTKTA MTXOB YNFGO GINUG JFNZV
QHYNG NEAJF HYOTW GOTHY NAFZN FTUIN ZBNEG NLNFU TXNXU
FNEJC INHYA ZGAEU TUCQG OGOTH JOHOA TCJXK HYNUV OCOHO
UHCNU GHHAF NUZHY NCUTW JUWNA EHYNA FOWOT UCHNP
HOGLN FQZNG FOUVC NVJHT AHNGG NTHOU CGJXY OGHYN ABNTO
TWGNT HNTXN AEBUF KNFYO HHGIU TJUCE AFHYN GACJH OATAE
IOCOH UFQXO BYNFG

如何解密以恢复明文？

解：(1) **先统计密文中各字母出现的次数**：(**由大到小排列**)

N(36)，H(26)，O(25)，G(23)，T(22)，U(20)，F(17)，A(16)，Y(14)，C(13)，J(12)，X(9)，E(7)，Z(7)，W(6)，B(5)，I(5)，Q(5)，V(4)，K(3)，L(2)，M(2)，P(1)

英文文章中的高概率字母 e、t、a、o、n、i、r、s、h **和中概率字母** d、l、u、c、m **之间的概率值有一明显的间断。由此我们大体可确定密文中出现频度较高的** 9 **个字符** N、H、O、G、T、U、F、A、Y **的对应范围，并且大体能肯定** N **就是** e。

(2) **再来统计密文中各字符的前缀与后缀的出现频度。下表列出了密文中出现频度较高的** 9 **个字符的前后连缀关系。表格中的每组数分别表示该行、该列两字符的两种不同排序方式所出现的次数。比如** N **行** G **列交叉处的**(5，4)**表示** NG **出现** 5 **次，**GN **出现** 4 **次。**

	N	H	O	G	T	U	F	A	Y	推测
N	0，0	1，3	0，0	5，4	4，0	5，0	7，3	5，0	0，9	e
H	3，1	2，2	4，5	1，2	1，4	1，1	0，2	1，1	10，0	t
O	0，0	5，5	0，0	10，6	7，1	1，0	1，1	2，0	0，3	i
G	4，5	2，1	6，4	2，2	0，0	0，2	0，2	2，0	0，0	?
T	0，4	4，1	1，7	0，0	0，0	3，4	0，1	3，2	0，0	n
U	0，5	1，1	0，1	2，0	4，3	3，2	0，0	0，0	0，0	a
F	3，7	2，0	1，1	3，0	1，0	2，3	0，0	0，3	1，0	?
A	0，5	1，1	0，2	0，2	2，3	0，0	4，0	0，0	0，1	o
Y	9，0	0，10	3，0	0，0	0，0	0，0	0，1	0，1	0，0	h

① 由这些数据推测，O、U、A 可能是元音 i、a、o，这是因为它们互连的可能很小；OA 出现两次，OU 出现一次，其他 UO、UA、AO、AU 均无出现，所以 O→i，A→O；又由 OU 出现一次，NU 出现 5 次，UN 不出现，可猜到 U→a(因 ea 常见 ae 极少，ia 也是存在的)。

② 根据 n 是高频辅音，且 n 的前面是元音的概率高达 80%，现在 T 频度为 22，前面是 O. U. A 的占 17 次，由此可判定 T→n。

③ YN 出现 9 次，NY 不出现，已经判知 N→e，于是可判 Y→h。根据 th 常见而 ht 罕见，及 HY 出现 10 次，YH 一次也没有，知 H→t。

④ 高概率集中只剩下 r 和 s 尚看不出与密文前 9 位中剩余的 F 和 G 有何对应关系。

(3) 接下来就把 280 个字母中已找到的 159 个先写出来，空出尚未找到对应关系的字母，待猜测填空(注意：大写表示密文字符，小写表示已经找到的明文字符)。

① 由 Mith 猜出 M→w，于是由 nKnown→nknown 知 K→k；由 intJition 知 J→u。

② 将已判定的字母填入置换对照表中就有：

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
U										K			N		Y	O			T	A			H	J	M

之后的 HJ-M 已呈现出字母表的顺序，于是可猜到中间空的是 L→v；而前面的 rs 应当是 FG，后面的 xy 则是 PQ；最前面既然 U→a，可猜到 V→b。

③ 再次将猜到的字母代入密文，就得到：

G J X X N G GO T Z N U C O T W M O H Y J T K T A M T X O B Y N F G

s uXXe s s i n Ze a Ci n Ww i t h unk n o w n ci Bh e r s

O G I N U G J F N Z V Q H Y N G N E A J FH Y O T W

i s Ie a s u r e Zb y t h e s e Eo u r t h i n W

由 suXXess 可猜出 X→c，于是由 XiBhers 知 B→p，得 ciphers；还可由 Eour 猜 E→f 得 four；thinW 猜 W→g，得 thing。

④ 再次回到单表对照关系上，就知密钥词组是：NEW YORK CITY。去掉重复的 Y，后面按顺序接字母表中尚未出现的字符，就是：NEW YORK CIT ABDFGHJLMPQS。最后将 UVXZ 循环移位到前面去，得到与字母表对应的置换密钥是 UVXZNEWYORKCIT-ABDFGHJLMPQS。

(4) 按此对照表，可将密文全文译出为

Success in dealing with unknown ciphers is measured by these four things in the order named, perseverance, careful methods of analysis, intuition, luck. The ability at least to read the language of the original text is very desirable but not essential. Such is the opening sentence of Parker Hitt's Manual for the Solution of Military Ciphers.

意思是：破译一未知密码是否成功，可由以下四个因素来衡量，按顺序为：毅力、审慎的分析方法、直观和运气。阅读原文文字的起码能力是需要的，然而不是必不可少的。这是 Parker Hitt 的“军事密码破译指南”一书的开场白。

1.3.2 对维吉利亚密码的分析[2,5]

维吉利亚密码属多表密码，明文相同的字符变换到密文中未必是相同的字符，只有当

这两个相同字符距离等于密钥长度时，它们对应的密文字符才会相同。因此分析密文中两个相同字符出现的位置，就是寻找密钥长度的关键。

1. 密钥长度的初步判断

首先统计密文中不同距离处出现相同字符的次数。例如密文是如下的280个字符：

UFQUIUDWFHGLZARIHWLLWYYFSYYQATJPFKMUXSSWWCSVFAEVW
WGQCMVVSWFKUTBLLGZFVITYOEIPASJWGGSJEPNSUETPTMPOPHZS
FDCXEPLZQWKDWFXWTHASPWIUOVSSSFKWWLCCEZWEUEHGVGLRL
LGWOFKWLUWSHEVWSWTTUARCWHWBVTGNITJRWWKCOYFGMILR
QESKWGYHAQENDIULKDHZIQASFMPRGWRVPBUIQQDSVMPFZMVEGE
EPFODJQCHZIUZZMXKZBGJOTZAXCCMUMRSSJW

从头到尾考察密文中所有的符号。统计距离为1的两个相同符号出现的次数，就是数出相邻两字符相同的次数，结果是20次；统计距离为2的两个相同符号出现的次数，也就是中间只隔一个符号的两相同符号的次数，结果共出现52次。再统计距离为3的两个相同符号出现的次数，得到是32次。下表列出了两个相同字符出现在距离20以内的各种情况的次数：

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
20	52	32	28	50	18	13	15	11	21	5	8	7	8	16	3	9	5	4	12

可以看到，距离为2和5时，两相同符号出现的次数最多，由此可以推测，密钥长度可能为2或5。但是长度为2的密钥极少有人取，于是初步判定密钥长度为5。

2. 密钥长度的确认

根据初步判断，可以把密文依次按列排序，每列5个字符，构成$m=5$行的阵列。也就是说，把原来的序列像发扑克牌似地依次分配到5行中：

UUGIWYJUWAGVUGTFGPTOFPKWPVKCUGGWHTCVTKGQGNKQPVQM
VPQUKOCR

FDLHYYPXCEQSTZYAGNPPDLDTWSWRELWLETWTJCMEYDDARPQPE
FCZZTCS

QWZWYQFSSVCWBFOSSSTHCZWHISWZHROUVUHGROISHIHSGBDFG
OHZBZMS

UFALFAKSVWMFLVEIJUMZXQFAUSLWGLFWWAWNWTLKAUZFWUSZ
EDZMGAUJ

IHRLSTMWFWVKLIIWEEPSEWXSOFCEVLKSSRBIWPRWELIMRIVMEJIX
JXMW

如果密钥长度确实是5，则若设密钥为$k=k_1k_2k_3k_4k_5$，那么阵列的第1行应当是明文中序号为1，6，11，16……的相应字符用第1位密钥k_1移位加密的结果，第2行是明文中序号为2，7，12，17……的相应字符用第2位密钥k_2移位加密的结果…… 由于同一行的各个元素都是由明文字符通过相同距离的移位产生的，而这种单表加密方式的字符替换不改变原来的概率分布，因此每行仍然应当呈现出与明文英语相同的统计规律。然而，如果

密钥长度本不是5，却按照上述方式排成了5行的阵列，则它的同一行各个元素是由不同的移位规律的多表加密方式产生，就不具备明文英语字符的统计规律，而呈现完全随机的统计特征。

完全随机排列的字符序列中，任一位置出现指定字符(比如A)的概率是$\frac{1}{26}$，在两位置都出现该指定字符的概率是$\left(\frac{1}{26}\right)^2$，所以两位置出现相同字符(不论什么字符)的概率是$26\times\left(\frac{1}{26}\right)^2=\frac{1}{26}=0.0385$；而如果是英语明文序列，不同字母出现的概率是不相同的，比如A是0.0856，两指定位置同时出现A的概率(假设为无记忆信源)是$(0.0856)^2$，B是0.0139，两指定位置同时出现B的概率(假设为无记忆信源)是$(0.0139)^2$…… 于是两指定位置出现任意相同字母的概率是$\sum_{\xi=A}^{Z}p_\xi^2=0.0687$。

设明文(密文)的长度为n，维吉利亚密钥长度为m。再设已统计出密文中字母A的数目为n_A个，B的数目为n_B个……，Z的数目为n_Z个，那么字母A构成的重复双A对(不论间隔多远)的数目是$\frac{1}{2}n_A(n_A-1)$，同理B构成的重复双B对的数目是$\frac{1}{2}n_B(n_B-1)$…… Z构成的重复双Z对的数目是$\frac{1}{2}n_Z(n_Z-1)$。所有相同字母对的数目为$S=\frac{1}{2}\sum_{\xi=A}^{Z}n_\xi(n_\xi-1)$。而长为$n$的密文中，任取两字符组成的对子的数目为$\frac{1}{2}n(n-1)$，可见两位置上字母相同的概率为

$$\mathrm{IC}=\sum_{\xi=A}^{Z}\frac{n_\xi(n_\xi-1)}{n(n-1)} \tag{1-11}$$

IC称为重合指数，表示指定序列中任意两位置上有相同字符的频度。

对上述阵列每行分别计算重合指数：第一行为0.0623，第二行为0.0506，第三行为0.0649，第四行为0.0617，第五行为0.0617；它们都很接近英文明文序列的统计结果0.0687，而明显不同于完全随机序列的理论值0.0385。这就表明了各行都是单表置换所得，也就是说，$m=5$的判断是正确的。

3. 密文的破译

我们不妨把一、二两行合起来作为一个整体来计算重合指数：

$$\mathrm{IC}=\sum_{\xi=A}^{A}\frac{(n_\xi^{(1)}+n_\xi^{(2)})(n_\xi^{(1)}+n_\xi^{(2)}-1)}{(n^{(1)}+n^{(2)})(n^{(1)}+n^{(2)}-1)}$$

式中，上角标(1)、(2)分别代表第一行和第二行，$n^{(1)}+n^{(2)}$则是两行总字符数。

因为

$$\begin{aligned}&\sum_{\xi=A}^{Z}(n_\xi^{(1)}+n_\xi^{(2)})(n_\xi^{(1)}+n_\xi^{(2)}-1)\\=&\sum n_\xi^{(1)2}+\sum n_\xi^{(2)2}+2\sum n_\xi^{(1)}\sum n_\xi^{(2)}-\sum n_\xi^{(1)}-\sum n_\xi^{(2)}\\=&\sum n_\xi^{(1)2}+\sum n_\xi^{(2)2}-2\sum n_\xi^{(1)}n_\xi^{(2)}-n^{(1)}-n^{(2)}\end{aligned}$$

式中，除 $n_\xi^{(1)}n_\xi^{(2)}$ 项外，其他都是常量，所以相关系数 $\sum_{\xi=A}^{Z}n_\xi^{(1)}n_\xi^{(2)}$ 最大，则 IC 最大。

设第一行的密钥为 k_1，第二行为 k_2，一般 $k_1\neq k_2$。我们首先计算密文一、二两行的相关系数 $X_{12}=\sum_{\xi=A}^{Z}n_\xi^{(1)}n_\xi^{(2)}$ 的值，之后将第一行各字符按字母表顺序右移一位后再计算第一行与第二行的相关系数 X_{12}，再将第一行各字符按字母表顺序右移二位再计算第一行与第二行的相关系数 X_{12}……如此作下去，这些 X_{12} 中最大的一个必定是右移 σ_{12} 位，使 $\sigma_{12}+k_1=k_2$ 的那个，因为它使得一、二两行都变成了相同密钥 k_2 产生的密文。

换言之，我们由 X_{12} 的最大，找到了一、二两行密钥序号之差 $\sigma_{12}=k_2-k_1$。同理，由计算一、三两行的 X_{13} 的最大值可找到一、三两行密钥序号之差 σ_{13}。同样还可找到 σ_{14} 和 σ_{15}。本例的结果是 $\sigma_{12}=9$，$\sigma_{13}=12$，$\sigma_{14}=16$，$\sigma_{15}=2$。

进行到这一步，其实已经有了解译密文的方法：把第二行各字符按字母顺序左移 9 位，把第三行各字符左移 12 位，第四行左移 16 位，第五行左移 2 位，于是所有密钥都变得与密钥一样了。之后把这样移位处理后的 5 段密文重新按原来的顺序连接起来，得到：

UWEEGUUKPFGCNKPIYKVJWPMPQYPEKRJGTUKUOGCUWTGFDAVJG
UGHQWTVJKPIUKPVJGQTFGTPCOGFRGTUGXGTCPEGECTGHWNOGV
JQFUQHCPCNAUKUKPVWKVKQPNWEMVUGCDKNKVACVNGCUVVQT
GCFVJGNCPIWCIGQHVJGQTKIKPCNVGZVKUXGTAFGUKTCDNGDWVP
QVGUUGPVKUNUWEJKUVJGQRGPKPIUGPVGPEGQHRCTMGTJKVVUO
CPWCNHQTVJGUQNWVKQPQHOKNKVCTAEKRJGTU

此文中所有的字符都来自同样的循环移位，只需找到这个位移值 k_1，问题就得到解决。对此单表密码问题，已有成熟的破译方法：统计各个字符出现的频度，不难得到，文中 G(序号为 6)出现最多，为 36 次，可以大体判定它就是 e(序号为 4)，因此位移值 $k_1=6-4=2$。将所有字符均按字母表顺序，移回 2 个字符，就得到与[例 1]内容完全相同的译文。

既然 $k_1=2$，那么就有 $k_2=11$，$k_3=14$，$k_4=18$ 和 $k_5=4$，表明密钥是“close”。

4. 破译方法的简化

实际上，由初步判断得到密钥长度 m，把密文按列排序写成 m 行的阵列后，分别统计出每一行($j=1,2,\cdots,5$)各字符出现频度分布矢量：

$$\boldsymbol{P}_j=\left\{\frac{n_A}{n},\frac{n_B}{n},\cdots,\frac{n_Z}{n}\right\}$$

就可以直接与常规英文字符频率(见表 1.4)矢量

$$\boldsymbol{Q}=\{0.0856,0.0139,0.0279,0.0378,0.1304,\cdots,0.0008\}$$

进行比较判定了，即将常规英文字符频率矢量进行各种距离($i=0,1,\cdots,25$)的循环移位，得到 26 个位移矢量 $\boldsymbol{Q}_i$，分别计算它们与第一行频度分布矢量的点乘积：

$$\boldsymbol{X}_{1Q}=\{\boldsymbol{P}_1\cdot\boldsymbol{Q}_0,\boldsymbol{P}_1\cdot\boldsymbol{Q}_1,\cdots,\boldsymbol{P}_1\cdot\boldsymbol{Q}_{25}\}$$

式中，$\boldsymbol{P}\cdot\boldsymbol{Q}=\sum_{i=A}^{Z}p_iq_i$。

显然，只有当位移值等于密钥 k_1 时，点乘积最大，由此就能得到 k_1。同样的，由第二行频度分布矢量与各个位移矢量点乘积的最大值可以得到 k_2。同理得到 k_3，k_4，k_5。知道

了密钥后，对密文实施反向移位的 Vigenere 加密变换，就可求得明文。

1.3.3 对 Hill 密码的分析

Hill 密码是 $\boldsymbol{x}=(x_1x_2\cdots x_m)\rightarrow\boldsymbol{y}=(y_1y_2\cdots y_m)$的线性变换，很难用唯密文攻击。在已知部分密文与对应明文的情况下可以用线性代数方法求出变换矩阵，即密钥。

【例 3】 已知明文＝Friday＝(5，17，8，3，0，24)，密文＝UNJVOU＝(20，13，9，21，14，20)，Hill 变换矩阵为 2×2 方阵，求此变换矩阵。

解：将 fr→UN 和 id→JV 代入变换公式 $\boldsymbol{c}=\boldsymbol{k}\cdot\boldsymbol{m}$，有：

$$\begin{pmatrix}20 & 9\\13 & 21\end{pmatrix}=\boldsymbol{k}\cdot\begin{pmatrix}5 & 8\\17 & 3\end{pmatrix}\bmod 26$$

$$\boldsymbol{k}=\begin{pmatrix}20 & 9\\13 & 21\end{pmatrix}\cdot\begin{pmatrix}5 & 8\\17 & 3\end{pmatrix}^{-1}=\begin{pmatrix}20 & 9\\13 & 21\end{pmatrix}\cdot\begin{pmatrix}9 & 2\\1 & 15\end{pmatrix}=\begin{pmatrix}7 & 19\\8 & 3\end{pmatrix}\bmod 26$$

以上是否正确呢？这可由 ay→OU 来验证：

$$\begin{pmatrix}7 & 19\\8 & 3\end{pmatrix}\begin{pmatrix}0\\24\end{pmatrix}=\begin{pmatrix}456\\72\end{pmatrix}\bmod 26=\begin{pmatrix}14\\20\end{pmatrix}\bmod 26$$

可见结果正确。

习 题 1

1. 凯撒密码密钥为 $k=11$，明文为 wewillmeetatmidnight，请将其加密。

2. 维吉尼亚密码的密钥字为“CIPHER”，试将明文“thiscryptosystemisnotsecure”加密。

3. 设密钥为 $\boldsymbol{k}=\begin{pmatrix}11 & 8\\3 & 7\end{pmatrix}$，明文为 july，用希尔加密算法加密，求密文。

4. 假设明文是 Breathtaking，使用 Hill 密码被加密为 RUPOTENTOIFV，试分析出加密密钥矩阵(矩阵维数 m 未知)。

5. 在 Alice 和 Bobe 的保密通信中，传送的密文是 Rjjy rj ts ymj xfggfym bj bnqq inxhzxx ymj uqfs，如果他们采用的是恺撒密码算法，试解密其通信内容。

实 践 练 习 1

实验题目：Vigenere 密文的生成与破译。

实验平台：Mathematica 4.0。

实验内容：编写程序将任意明文加密成 Vigenere 密文，然后学生之间互相交换密文后进行破译训练。

实验步骤：(1) 编写 Vigenere 加密程序。

(2) 对收到的密文统计不同距离处出现相同字符的次数，初步确定密钥长度 m。

（3）将密文写成 m 列的阵列。

（4）写出常规英语字符频率矢量及其各个循环移位矢量 $\boldsymbol{Q}_i$。

（5）分别计算密文阵列各行字符出现频度分布矢量 $\boldsymbol{P}_j$ 与各个常规英语字符频率移位矢量 $\boldsymbol{Q}_i$ 的点乘积 $\boldsymbol{X}_{ji}$，并由各行 $\boldsymbol{X}_{ji}$ 的最大值确定密钥的相应值 k_i。

（6）用 Vigenere 加密程序验证破译结果。（用密钥的负值加密就是解密）

第2章 序列密码

把明文加密成密文，有两种方式，一种是分组(块)加密，一种是序列(流)加密。采用序列加密时，明文不进行分段，明文序列被源源不断地送入加密器，在密钥作用下进行加密运算，产生的密文序列源源不断地被输出来，实时地完成加密过程。本章主要讨论伪随机序列的产生和特点，因为它不仅是序列加密密钥的来源，而且也是现代密码体制中不可缺少的关键数据。

2.1 序列密码原理

2.1.1 序列加、解密原理

一个序列流加密的保密通信系统的流程如图 2.1 所示。

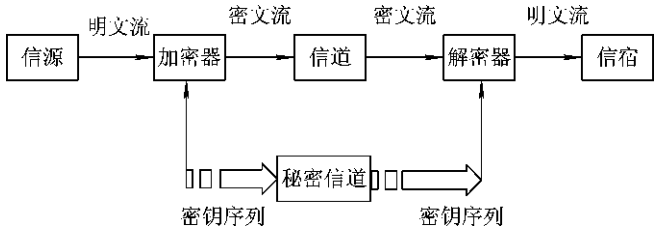

图 2.1 序列流加密的保密通信系统流程图

明文消息序列为

$$m = m_1 m_2 m_3 \cdots$$

密钥序列为

$$k = k_1 k_2 k_3 \cdots$$

加、解密都是逐位顺序进行的：

$$c_i = E_k(m_i)；\ m_i = D_k(c_i) \tag{2-1}$$

得到的密文序列为

$$c = c_1 c_2 c_3 \cdots$$

2.1.2 安全性讨论

从传统密码算法得知，不论单表加密、多表加密，或是置换还是其他方式，密文之所以“密”，原因在于密钥使明文变“乱”了，这种乱就是随机性。密钥的随机性掩盖了明文的可理解性。

Shannon 信息论已经证明[1]，只有“一字一密”系统才是不可破译的、完全安全的保密系统。所谓一字一密，指密钥序列是无限长的真正的随机序列，密钥使每位密文都变得无规律可循。对于这样的系统，破译者完全无法从已有的密文中，或明、密文对照中找到规律，不能为未知的密文破译提供任何有价值的信息。

然而，这样的保密系统是不可行的。且不说真正的随机序列如何获得，单就如何把无限长的密钥安全地传给合法收信人就是个难题。如果密钥和密文一样长，用秘密信道传递密钥还不如直接传递明文好了！保密通信的初衷就是希望在公开信道中安全通信，因为秘密信道实在昂贵而又难得。

有实际意义的密钥应当是用软、硬件可以再生的“伪”随机序列，伪就伪在它并不真正随机，而是有规律的，这个规律就是周期性。只不过周期很长，在有限时段内看不出规律罢了。生成这样的伪随机序列，只需要很短的“初始密钥”(或曰种子)，只要通信双方在协议中约定好伪随机序列的产生方法和初始值，即具有了共同的密钥。图 2.2 为用伪随机序列作为密钥的序列流加密系统。

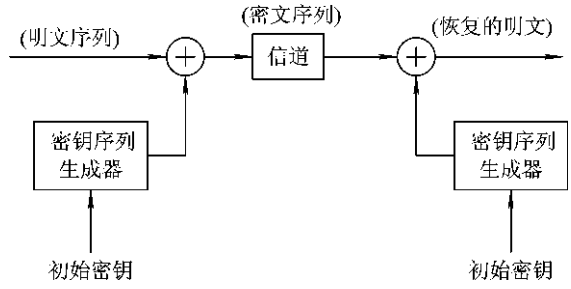

图 2.2 用伪随机序列作为密钥的序列流加密系统

2.1.3 序列密码对密钥流的要求[6]

1. 极大的周期

现代密码机数据率高达 10^8 bit/s。如果使用 10 年内不会重复的密钥流，密钥的周期应不少于 3×10^{16} 或 2^{55}。

2. 良好的统计特性

Golomb 对随机序列的统计性质提出了三条要求：

(1) 在序列的一个周期内，0 和 1 的个数最多相差 1。

(2) 在一个周期圈内，长度为 1 的游程数占 1/2，长度为 2 的游程数占 1/4…… 长度为 i 的游程数占 $1/2^i$，且在等长的游程中，0 游程和 1 游程各半(游程指连 0 或连 1 的段落)。

(3) 自相关函数为二值函数：

$$R_x(\tau)=\begin{cases}0 & \text{当 } \tau\neq 0 \text{ 时}\\ 1 & \text{当 } \tau=0 \text{ 时}\end{cases} \tag{2-2}$$

伪随机序列的自相关函数定义如下：

$$R_a(\tau)=\frac{1}{T}\sum_{k=0}^{T-1}(-1)^{a_k}\cdot(-1)^{a_{k+\tau}} \qquad 0\leqslant\tau\leqslant T-1 \tag{2-3}$$

式中，T 为周期；τ 为预先指定的一个位移值。相关函数定义为序列与它的移位序列逐位相

比较，相同则为 1，不同则得−1，然后相加起来求平均。因此，$R_a(\tau)$也等于两序列逐位相比较时，取值相同的位数与取值不同的位数之差再除以 T：

$$R(\tau) = \frac{n_\tau - d_\tau}{T}$$

对于上式，当 $\tau \neq 0$ 时 $R_a(\tau) \equiv 0$，表明通过$\{a_i\}$与$\{a_{i+\tau}\}$比较，无法得到关于$\{a_i\}$的实质性信息(比如周期)；而当 $\tau = 0$ 时，$R_a(0) = 1$，表明自身归一。

3. 足够的复杂度

用较短的“种子”就能生成周期较长的伪随机密钥，那么破译者也可以这么做，他只需掌握(破译或猜出)这个种子即可。因为产生伪随机序列的算法是公开的(至少是不难获得的)。因此，为了系统的安全，仅仅周期长是不够的，还要求密钥序列有足够的复杂度。

序列的复杂度指序列有足够繁多的变化花样。它的具体定义后面会讨论。现在要说明的是，伪随机序列生成器输出的虽然不是真正的随机序列，但是在计算资源受到一定限制(速度、内存有限，机时有限)的条件下，如果要想区别这个输出序列与相同长度的真随机序列，是不可行的；或者从理论上可以证明，二者的计算工作量随序列长度 n 变化的依赖关系以比多项式或更高(如幂指数函数)的方式而增大，因而可以认为二者是不可区分的。这样的伪随机序列就被认为具有足够的复杂性。

说到底，世上没有绝对不可破的密码，比如通过穷搜索总能将其破译，然而如果穷搜索不可能在有限时间内完成，而且又不存在比穷搜索更好的破译方法，则可以认为这个密码系统是安全的。

2.2 线性反馈移位寄存器[3]

20 世纪 50 年代以后，数字电子技术大力发展，使得伪随机序列可以方便地利用移位寄存器产生，加上有效的数学工具(域理论和谱分析)，使序列密码迅速走向成熟。

2.2.1 反馈移位寄存器(Feedback Shift Register)

n 位二值存储器共有 2^n 种状态，每个状态都代表一个长度为 n 的序列$(a_0 a_1 a_2 \cdots a_{n-1})$。在主时钟脉冲作用下，寄存器各位共同右移一位，$a_0$ 输出，而 a_{n-1}来自原先状态所决定的一个反馈值 $f(a_0\ a_1\ a_2 \cdots a_{n-1})$。$f(a_0\ a_1\ a_2 \cdots a_{n-1})$是预先设定好的一个 n 元布尔函数，于是有关系：

$$\begin{cases} a_i(t+1) = a_{i+1}(t) & (i = 0, 1, 2, \cdots, n-2) \\ a_{n-1}(t+1) = f(a_0(t), a_1(t), \cdots, a_{n-1}(t)) \end{cases} \tag{2-4}$$

特别是当 $f(a_0\ a_1\ a_2 \cdots a_{n-1})$为线性函数时，有

$$f(a_0\ a_1\ a_2 \cdots a_{n-1}) = c_0\ a_0 \oplus c_1\ a_1 \oplus \cdots \oplus c_{n-2}\ a_{n-2} \oplus c_{n-1} a_{n-1} \tag{2-5}$$

这里，各 $c_i \in \mathrm{GF}(2)$，即：

$$c_i = \begin{cases} 0 & \text{表示该位无反馈} \\ 1 & \text{表示该位有反馈} \end{cases} \tag{2-6}$$

这个系统称为线性反馈移位寄存器(LFSR)，如图 2.3 所示。

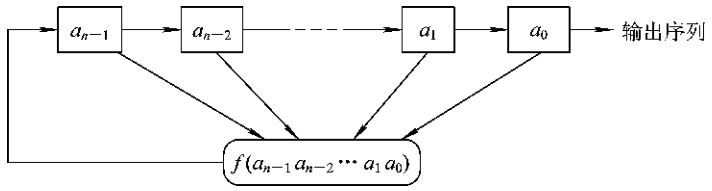

图 2.3　反馈移位寄存器

如图 2.4 所示的三级线性反馈移位电路，任意给定非零初态就能输出一个周期为 $T=7$ 的伪随机序列。图 2.4 中，左右两电路都是三级线性反馈移位器，区别仅在于抽头的位置不同。当初态为 001 时，左右两电路寄存器状态的变化如表 2.1 所示。它们分别不断循环地输出 1001011 和 1001110。

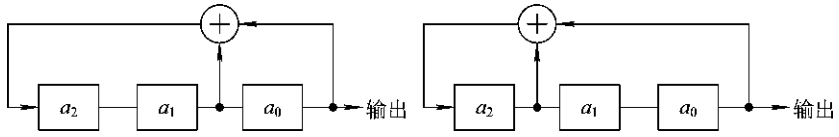

图 2.4　三级移位寄存器构成的 m 序列发生器

表 2.1　三级 m 序列的输出

时序	左图	右图	时序	左图	右图
1	001	001	5	110	011
2	100	100	6	111	101
3	010	110	7	011	010
4	101	111	8	001	001

$f(a_0a_1\cdots a_{n-1})$也可以是非线性函数，如：

$$f(a_0a_1a_2)=a_0\oplus a_1a_2$$

当初态为 101 时，可得到周期为 4 的伪随机序列 1011 的循环，它的电路如图 2.5 所示。

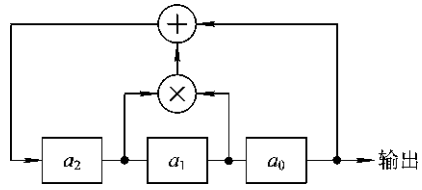

图 2.5　非线性反馈三级移位寄存器

2.2.2　m 序列

1. 特征多项式

为了利用多项式的代数理论来研究线性反馈移位寄存器(LFSR)产生伪随机序列的规律，定义特征多项式：

$$p(x)=1+c_1x+c_2x^2+\cdots+c_{n-1}x^{n-1}+c_nx^n=\sum_{i=0}^{n}c_ix^i \qquad (2-7)$$

用它来描述线性反馈移位寄存器的构造：各 x^i 只是为了区分寄存器的位序号，而各 c_i 值表示该位有无反馈，有则为 1，无则为 0。

为什么会是一个 n 次多项式呢？因为 LFSR 各位的输入都来自它前一位的输出，x^{n-1} 位的输入来自 x^n 位的输出，而 x^n 位的寄存器是不存在的，代替 x^n 位的是反馈回来的数据，反馈关系可表达为

$$x^n = 1 \oplus c_1 x \oplus c_2 x^2 \oplus \cdots \oplus c_{n-1} x^{n-1} \tag{2-8}$$

将等号左面的 x^n 移到等号右面，便是特征多项式 $p(x)$。

图 2.4 给出的两个三级线性反馈移位电路的特征多项式分别为

$$p(x) = x^3 + x + 1 \quad \text{和} \quad p(x) = x^3 + x^2 + 1$$

它们对应的本原元素 α 满足方程 $p(\alpha)=0$，在模 2 运算下等价于：

$$\alpha^3 = \alpha \oplus 1 \quad \text{和} \quad \alpha^3 = \alpha^2 \oplus 1$$

此即电路抽头位置所表示的反馈关系。

当然还可以存在其他形式的三级线性反馈移位电路，如：

$$p(x) = x^3 + 1 \quad \text{和} \quad p(x) = x^3 + x^2 + x + 1$$

但它们产生的分别是周期为 $T=3$ 和 $T=4$ 的伪随机序列，其电路如图 2.6 所示。

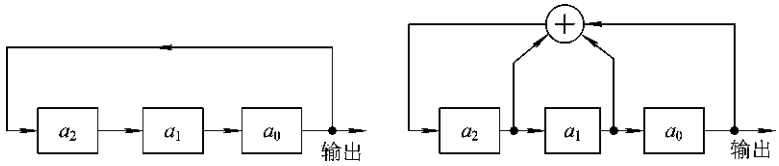

图 2.6　三级移位寄存器构成的 m 序列发生器

写出一个特征多项式 $p_n(x)$，就能构造出一个线性移位寄存器，于是就对应着一个伪随机序列$\{a_i\}$，因而我们常用$\{a_i\} \in S\{p_n(x)\}$表示$\{a_i\}$属于由 $p_n(x)$ 所描述的非零伪随机序列。这里的 n 表示 n 级反馈移位寄存器。

2. *m* 序列

三级的线性反馈移位寄存电路所产生的伪随机序列，其周期最长只能为 7。这是因为 $n=3$ 位的寄存器全部状态共 $2^n=8$ 个，除全零状态之外，最多只能在 $2^n-1=7$ 个状态间循环。所以，n 级的线性反馈移位寄存电路所产生的伪随机序列，其周期 T 不会大于 2^n-1。

定义：周期达到 2^n-1 的 n 级线性反馈移位寄存器的输出序列，称为 m 序列。

定理：$\{a_i\}$为 m 序列的充要条件是 $p_n(x)$为本原多项式。

事实上，$p_n(x)$是本原多项式，就表明 $p_n(x)$的根是 GF(2^n)域的本原元素，而本原元素的循环阶是 $T=2^n-1$，达到了最大的循环周期。

因此，要想知道 n 级线性反馈移位寄存器能产生多少种不同的 m 序列，只要看GF(2^n)域中 n 级的本原多项式有多少个即可。根据有限域理论，计算公式是：

$$\lambda(n) = \frac{\Phi(2^n - 1)}{n} \tag{2-9}$$

式中，$\Phi(T)$为欧拉数，代表与 T 互素的同余类的个数(凡是除以 T 后余数相同的整数为一个同余类)。$\Phi(T)$的计算公式是：

$$\Phi(T)=\begin{cases}T-1 & \text{当 } T \text{ 为素数时}\\ T\left(1-\dfrac{1}{p_1}\right)\left(1-\dfrac{1}{p_2}\right)\cdots\left(1-\dfrac{1}{p_k}\right) & \text{当 } T \text{ 为合数时，即 } T=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}\end{cases} \tag{2-10}$$

例如，对于 $n=3$ 的 LFSR，有

$$T=2^3-1=7;\ \Phi(7)=7-1=6;\ \lambda(n)=\frac{6}{3}=2$$

表明存在两个 3 次本原多项式，分别为

$$p(x)=x^3+x+1 \quad \text{和} \quad p(x)=x^3+x^2+1$$

分别产生两种不同抽头位置的循环移位电路。

又如，$n=4$ 时，

$$T=2^4-1=15;$$

$$\Phi(15)=15\times\left(1-\frac{1}{3}\right)\times\left(1-\frac{1}{5}\right)=2\times4=8;$$

$$\lambda(n)=\frac{8}{4}=2$$

两个 4 次本原多项式是：

$$p(x)=x^4+x+1 \quad \text{和} \quad p(x)=x^4+x^3+1$$

表 2.2 中列出了 $n\leqslant24$ 以内的线性反馈移位寄存器的周期和 m 序列的个数。

表 2.2　$n=24$ 以内的线性反馈移位寄存器的周期和 m 序列的个数

n	$T=2^n-1$	$\lambda(n)$	n	$T=2^n-1$	$\lambda(n)$
1	1	1	13	8191	630
2	3	1	14	16 383	756
3	7	2	15	32 767	1800
4	15	2	16	65 535	2048
5	31	6	17	131 071	7710
6	63	6	18	262 143	7776
7	127	18	19	524 287	27 594
8	255	16	20	1 048 575	24 000
9	511	48	21	2 097 151	84 672
10	1023	60	22	4 194 303	120 032
11	2047	176	23	8 388 607	356 960
12	4095	144	24	16 777 215	276 480

【例 1】　构造一个 $n=5$ 的 LFSR，并计算所产生 m 序列的周期和个数。

解：$n=5$ 级的线性反馈移位寄存器所产生的 m 序列的周期为

$$T=2^5-1=31$$

由

$$\Phi(31)=31-1=30;\ \lambda(n)=\frac{30}{5}=6$$

知共有 6 种不同的五级 m 序列。

不妨查本原多项式表，找到 $n=5$ 的一个本原多项式，其八进制表达形式为

$$(45)_8 = (100101)_2$$

即

$$p(x)=x^5+x^2+1$$

对应的电路如图 2.7 所示。

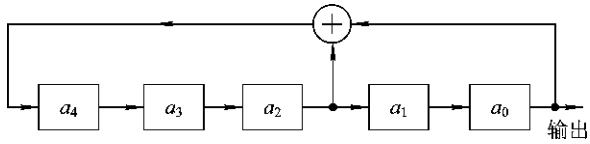

图 2.7　$n=5$ 的移位寄存器构成的 m 序列发生器

2.2.3　线性反馈移位寄存器的软件实现

m 序列也可以用计算机程序产生。首先根据所需伪随机序列周期来设计移位寄存器的级数 n，然后从本原多项式表中选取 $p_n(x)$。下面是一些本原多项式的非零系数：

(3, 1, 0), (4, 1, 0), (5, 2, 0), (6, 1, 0), (7, 3, 0), (9, 4, 0), (10, 3, 0), (11, 2, 0), (15, 1, 0), (17, 3, 0), (17, 6, 0), (18, 7, 0), (20, 3, 0), (21, 2, 0), (23, 5, 0), (28, 3, 0), (29, 2, 0), (31, 3, 0), (32,7,6,2,0), (32.7, 5, 3, 2,1,0), (33, 16, 4, 1, 0), …

以

$$p_n(x)=x^{32}+x^7+x^5+x^3+x^2+x+1$$

为例，根据特征多项式不难画出一个 32 位的线性反馈移位寄存器电路示意图，如图 2.8 所示。

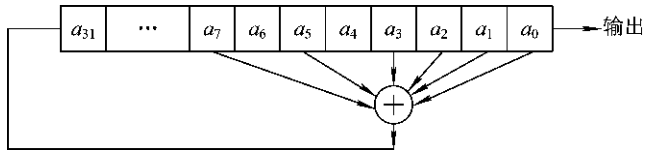

图 2.8　$p_n(x)=x^{32}+x^7+x^5+x^3+x^2+x+1$ 的电路

C 语言源程序为

```
Int LFSR( ) {
    Static unsigned long
    Shiftregister=1;
    / * Anything but 0. * /
    Shiftregister=((((shiftregister>>7)^(shiftregister>>5)
            ^(shiftregister>>3)^(shiftregister>>2)
            ^(shiftregister>>1)^shiftregister)&0x00000001)<<31)|(shiftregister>>1);
    return shiftregister&0x00000001;
}
```

2.2.4　m 序列的统计性质

m 序列的统计性质如下：

(1) 在一个周期内 0 和 1 的个数至多相差 1 位。

(2) 游程特征基本符合 Golomb 要求：

① 任一循环周期中含 2^{n-1} 个 1 和 $2^{n-1}-1$ 个 0。这是因为序列的排布完全取决于寄存器中状态的变化，每当 a_0 移出一位成为序列中新加入的一个码元后，寄存器上一位又移入 a_0，可见 a_0 有过些什么值，输出序列就出现些什么值。n 位寄存器共 2^n 种不同的状态，a_0 位为 1 与为 0 各占 2^n 种的一半，即 2^{n-1} 次，所以 m 序列的任一循环周期中 1 出现 2^{n-1} 次；由于全 0 状态不出现，因而 0 出现 $2^{n-1}-1$ 次。

② 游程最长为 n，有一个长为 n 的 1 游程，却没有长为 n 的 0 游程。这是因为只有一个全 1 状态，而没有全 0 状态。

③ 没有长为 $n-1$ 的 1 游程，却有一个长为 $n-1$ 的 0 游程。这是因为已经有了一个全 1 状态，它的前一个状态是 11…110，后一个状态是 011…11，二者虽都是 $n-1$ 个 1 的状态，却不能产生 $n-1$ 的 1 游程，因为这三个状态进入系列后产生的是同一个长为 n 的 1 游程。而这两个 $n-1$ 个 1 的状态既然已经出现，同一周期中就不会再出现第三个 $n-1$ 个 1 的状态了。长为 $n-1$ 的 0 游程却是可以存在的，这是由于不存在全 0 状态，所以 $n-1$ 个 0 的状态 100…00 和 00…001 都可以存在，它们产生的是同一个长为 $n-1$ 的 0 游程。

④ 长为 $r\leqslant n-2$ 的 1 游程和 0 游程各为 2^{n-r-2} 个。这是因为这种游程意味着状态中存在形如 011…110(r 个 1)的串，这种游程的数目就等于状态中其余 $n-r-2$ 位的状态数目，此即 2^{n-r-2}。

⑤ 每一循环中共有 2^{n-2} 个 1 游程和 2^{n-2} 个 0 游程(长度大于 1 的)。这是因为：

$$1+\sum_{r=1}^{n-2}2^{n-r-2}=2^{n-2}$$

(3) 周期为 2^n-1 的 m 序列，其异相自相关函数等于 $\dfrac{-1}{2^n-1}$。

证明：由前知 $R(\tau)$ 等于 $\{a_i\}$ 和 $\{a_{i+\tau}\}$ 两序列对应位置上有相同值的个数减去不同值的个数，再除以周期 T。当 $\{a_i\}$ 与 $\{a_{i+\tau}\}$ 逐位相加后，得到的必然仍为 $S\{P_n(x)\}$ 中的一个序列 $\{b_j\}$。$\{b_j\}$ 中的 0 代表 $\{a_i\}$ 和 $\{a_{i+\tau}\}$ 对应位相同的情况，$\{b_j\}$ 中的 1 代表 $\{a_i\}$ 和 $\{a_{i+\tau}\}$ 中对应位不同的情况，而由性质①知道，0 的个数比 1 的个数少一个，故分子是 -1，分母是周期 2^n-1。

(4) n 级线性移位寄存器可以产生 $\lambda(n)$ 种不同结构的 m 序列；每给定一种结构，还可以选择 2^n-1 个不同的初始状态，因而所能产生密钥的总数是 $\lambda(n)\cdot(2^n-1)$ 个。

2.2.5 线性反馈移位寄存器输出序列的复杂度

能用线性反馈移位寄存器来产生有限长度的任意序列 $\{a_i\}$ 吗？设 $\{a_i\}=a_0a_1a_2\cdots a_{N-1}$，序列长度为 N。显然，长度为 N 的自然码中肯定包含它，因此 N 级 LFSR 肯定可以产生它，只要取反馈函数 $f(x)=1+x^N$，初始状态是什么自然码就产生什么序列。然而这并不是最小的 LFSR。

任意给定一个长度为 N 的伪随机序列 $\{a_i\}$，如何构造一个尽可能短的 LFSR 来产生它？可以分三种情况来讨论：

(1) 存在一个 n，恰能使 $N=2^n-1$，并且序列 $\{a_i\}$ 的排列情况在 $\lambda(n)$ 种周期为 N 的 m 序列中可以找到，则相应的 n 级(LFSR)就是能产生 $\{a_i\}$ 的最短的发生器。

(2) 虽然 $N=2^n-1$，但因所给定$\{a_i\}$的排布花样更多，变化更复杂，以致于在 $N=2^n-1$ 的 $\lambda(n)$个 m 序列中不能找到它，需要在更大的 n 值(比如 $n_1>n$)的 m 序列中才能包括它，那么就可以用级数为 n_1 的 LFSR 来产生它。显然，这时 $N_1=2^{n_1}-1$，使 $N_1>N$，$\{a_i\}$只是周期为 N_1 的 m 序列中的一个段落。

(3) 不存在恰能使 $N=2^n-1$ 的 n 值，但可以找到一个尽可能小的 L 值，使 $N<2^L-1$，并且序列$\{a_i\}$在 $\lambda(L)$种周期为 2^L-1 的 m 序列中可以找到，或者被包含，那么就可以用级数为 L 的 LFSR 产生它。

以上分析表明，序列越复杂，恰能产生它的 m 序列的级数就越高，因此要定义任意一个伪随机序列$\{a_i\}$的复杂度 $C(\{a_i\})$，可以先来寻找恰能包含它的最短的 m 序列，然后用产生这个 m 序列的 LFSR 的级数 n 来作为它的复杂度的定义。

定义：二元序列$\{a_i\}=a_0a_1\cdots a_{N-1}$的线性复杂度 $C(\{a_i\})$的定义是恰能产生完整包含该序列$\{a_i\}$的级数最少的线性反馈移位寄存器的级数 n。

$$C(\{a_i\})=n_{\mathrm{LFSR}} \tag{2-11}$$

【例 2】 讨论下述 3 个 $N=6$ 的序列的复杂度。

(1) $\{a_i\}=100101$；

(2) $\{b_i\}=100110$；

(3) $\{c_i\}=101010$。

解：(1) $\{a_i\}=100101$：它被包含在三级 LFSR 产生的序列//1001011//中，所以它的复杂度为 3；

(2) $\{b_i\}=100110$：它也是 $N=6$ 的序列，但在两个三级 LFSR 序列//1001011//和//1001110//中都找不到，在四级 LFSR 的 m 序列//1000 10011010111//中方可包含，因此它的复杂度为 4。

(3) $\{c_i\}=101010$：它还是 $N=6$ 的序列，但却只能在五级 LFSR 的 m 序列//10000100101100111111000110111 1010//中才能找到相应段落，因此它的复杂度 $C(\{a_i\})=5$。

对全零序列$\{a_i\}$约定 $C(\{a_i\})=0$；长度为 N 的任意序列，复杂度最大为 N；如果是 n 级 m 序列，则复杂度为

$$C(\{a_i\})=n=\mathrm{lb}(N+1)$$

显然，用线性反馈移位寄存器产生的伪随机序列，其复杂度都不高。由表 2.2 中所列数据看到，周期为 $T=16\ 777\ 215$ 的序列，复杂度才是 $C(\{a_i\})=n=24$。

结论：线性反馈移位寄存器产生的 m 序列，是构造密钥序列的好素材，其周期长度可以做到很大，随机性也十分接近 Golomb 要求。但缺点是复杂度太低，必须设法解决。

2.2.6 m 序列的破译[5]

作为密钥用的伪随机序列，复杂度过低有何问题呢？答案是安全性太低！理论上可以证明，只要知道 n 级 m 序列中的相继 $2n$ 位，就能推测出它的特征多项式，进而得到整个序列。

由式(2-5)知，n 级线性反馈移位寄存器的反馈关系可表示为

$$c_0a_0 \oplus c_1a_1 \oplus \cdots \oplus c_{n-2}a_{n-2} \oplus c_{n-1}a_{n-1}=a_n \tag{2-12}$$

因为 m 序列实际上是线性反馈移位寄存器历史状态的记录，所以线性反馈移位寄存器状态的反馈关系也就是 m 序列相邻 $n+1$ 位之间的约束关系。设 m 序列的连续 $2n$ 位为

$$x_1,\ x_2,\ x_3,\ \cdots,\ x_n,\ x_{n+1},\ \cdots,\ x_{2n}$$

则相邻 $n+1$ 位之间有以下的递推关系：

$$\begin{aligned}
&c_0x_1 \oplus c_1x_2 \oplus \cdots \oplus c_{n-2}x_{n-1} \oplus c_{n-1}x_n = x_{n+1}\\
&c_0x_2 \oplus c_1x_3 \oplus \cdots \oplus c_{n-2}x_n \oplus c_{n-1}x_{n+1} = x_{n+2}\\
&c_0x_3 \oplus c_1\ x_4 \oplus \cdots \oplus c_{n-2}x_{n+1} \oplus c_{n-1}x_{n+2} = x_{n+3}\\
&\qquad\vdots\\
&c_0x_n \oplus c_1x_{n+1} \oplus \cdots \oplus c_{n-2}x_{2n-2} \oplus c_{n-1}x_{2n-1} = x_{2n}
\end{aligned}$$

写成矩阵形式为

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n\\ x_2 & x_3 & \cdots & x_{n+1}\\ & & \vdots & \\ x_n & x_{n+1} & \cdots & x_{2n-1}\end{pmatrix}\begin{pmatrix} c_0\\ c_1\\ \vdots\\ c_{n-1}\end{pmatrix}=\begin{pmatrix} x_{n+1}\\ x_{n+2}\\ \vdots\\ x_{2n}\end{pmatrix} \tag{2-13}$$

如果上式左面的 $n\times n$ 方阵（不妨叫做错位方阵）是非奇异的，逆矩阵存在，就能够得到：

$$\begin{pmatrix} c_0\\ c_1\\ \vdots\\ c_{n-1}\end{pmatrix}=\begin{pmatrix} x_1 & x_2 & \cdots & x_n\\ x_2 & x_3 & \cdots & x_{n+1}\\ & & \vdots & \\ x_n & x_{n+1} & \cdots & x_{2n-1}\end{pmatrix}^{-1}\cdot\begin{pmatrix} x_{n+1}\\ x_{n+2}\\ \vdots\\ x_{2n}\end{pmatrix} \tag{2-14}$$

这样，就由相继 $2n$ 位 m 序列求出了各个反馈系数。

然而，一般情况下总是只知道一段连续的 m 序列，并不知道产生它的 LFSR 的级数 n，破译者首先必须设法确定 n 值是多少，才能用式(2-14)来计算各反馈系数。

设已知的 m 序列为 x_1，x_2，x_3，…；因为我们不知道 n 等于多少，不妨从 $k=1$，2，3，…测试起，这时式(2-13)的左边的方阵分别为

$$(x_1);\ \begin{pmatrix} x_1 & x_2\\ x_2 & x_3\end{pmatrix};\ \begin{pmatrix} x_1 & x_2 & x_3\\ x_2 & x_3 & x_4\\ x_3 & x_4 & x_5\end{pmatrix};\ \cdots;\ \begin{pmatrix} x_1 & x_2 & \cdots & x_k\\ x_2 & x_3 & \cdots & x_{k+1}\\ & & \vdots & \\ x_k & x_{k+1} & \cdots & x_{2k-1}\end{pmatrix}\cdots \tag{2-15}$$

对每一种 $k\times k$ 方阵，首先计算模 2 行列式是否为零，若为零，则逆矩阵不存在，表明所测试的 k 值不对。若行列式非零，则由式(2-14)解出相应的系数 c_0，c_1，…，c_k，把这些系数代回式(2-13)，检验后面的所有数据是否满足。一旦发现有不满足的数据，则说明所测试的 k 值还是不对。只有后面的所有数据统统满足式(2-13)，才表明所测试的 k 值是 m 序列的级数 n。当测试通不过时，应再测试更大的方阵。

一种简化的测试方法是，计算式(2-15)的各个行列式模 2 值，凡是等于 0 的都可以剔除，在模 2 值等于 1 的行列式中，k 值最大的那个矩阵就应当是所寻找的矩阵。将其代回式(2-14)即可求得各个反馈系数。

2.3 非线性序列[6]

2.3.1 M 序列

如果反馈函数 $f(a_1a_2\cdots a_n)$ 为非线性函数，便构成非线性移位寄存器，其输出序列为非线性序列，其周期最大可以达到 2^n。能达到这个最大周期 2^n 的序列叫做 M 序列。

M 序列具有以下统计特性：

(1) 在一个周期(2^n 位)之内，0 和 1 的个数各为一半(2^{n-1})。

(2) 在一个周期内，总游程为 2^{n-1}，其中 0、1 游程各半：长为 n 的 0 游程和 1 游程各一个，长为 $n-1$ 的 0、1 游程均不存在，长为 $i(1\leqslant i\leqslant n-2)$ 的 0、1 游程各 2^{n-i-1} 个。

(3) 周期为 2^n 的序列共有 2^{2^n} 个，但它们并非都是 M 序列，其中 n 级线性反馈寄存器产生的有 2^n 种，由它们生成(不同初态)的线性序列就有 $2^n\times 2^n=2^{2n}$ 种。理论指出在GF(2)域上，n 级 M 序列的总数为 $2^{2^{n-1}-n}$ 个，当 n 较大时，非线性反馈系列的数量是巨大的，有很大的利用空间。

(4) n 级 M 序列的复杂度为

$$2^{2^{n-1}}+n\leqslant C(\{a\})\leqslant 2^n-1,\ C(\{a\})\neq 2^{n-1}+n+1$$

由上述性质可见，M 序列有很好的统计性质，较高的复杂度和大量可供选择的序列，然而，并不是所有的 M 序列都能当作良好的密钥序列使用。鉴于非线性函数的复杂性，目前对非线性移位寄存器的研究仍处于非常艰难的地步，取得成果较多的是通过线性移位寄存器来开发的非线性序列。

2.3.2 非线性前馈序列

线性反馈移位寄存器产生的 m 序列，因其复杂度太低而不便于直接作为密钥流使用，但可以用它作为一个驱动源来推动另一个非线性电路。图 2.9 所示的 $f(x)$ 是非线性网络，也叫前馈电路，当 LFSR 处于不同状态时，$f(x)$ 便通过各位之间的非线性运算，输出不同的 k_i 值。

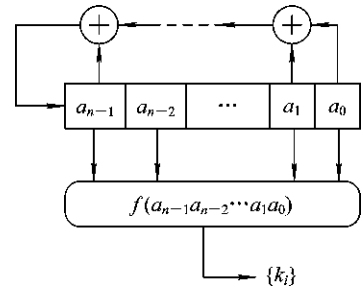

图 2.9 非线性前馈序列电路原理图

显然，$\{k_i\}$ 序列的周期与原来 LFSR 的周期相同，然而复杂度却大大提高了。$\{k_i\}$ 称为前馈序列。

理论证明，如果 $f(x)$ 的次数为 l，那么由 n 级 m 序列为驱动源的前馈序列 $\{k_i\}$ 的复杂度满足：

$$C(\{k_i\})\leqslant \sum_{j=1}^{l} C_n^j \tag{2-16}$$

式中，C_n^j 是 n 中取 j 的组合数。

由于前馈序列的周期等于 LFSR 的周期 $T=2^n-1$，因此由前馈电路输出的非线性伪

随机序列周期也是 T，可知它的复杂度最大是 T。

一般来说，通过前馈网络适当的非线性滤波，可以得到复杂度接近 2^n-1 的前馈序列。前馈序列的随机性与前馈网络的设计有关，比如增加项数可以改善前馈序列的统计特性。

2.3.3　非线性组合序列

用多个线性反馈移位序列共同驱动一个非线性网络，产生的非线性序列(见图 2.10)具有更好的特性：

(1) 若各个 LFSR 的级数 n_i 两两互素，则组合序列的周期为

$$T=\prod_{i=1}^{n}(2^{n_i}-1) \tag{2-17}$$

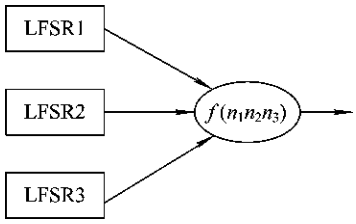

图 2.10　多个线性反馈移位序列驱动的非线性网络

(2) 组合序列的复杂度为

$$C(\{k_i\})=f(n_1n_2\cdots) \tag{2-18}$$

例如，3 个 LFSR 的级数分别为 3、5、7，非线性函数为

$$f(x_1,\ x_2,\ x_3)=x_3+x_1x_2+x_1x_2x_3$$

则组合序列的周期为

$$(2^3-1)(2^5-1)(2^7-1)=7\times31\times127=27\ 559$$

复杂度为

$$f(n_1,\ n_2,\ n_3)=7+3\times5+3\times5\times7=127$$

1973 年，P. R. Geffe 提出一种组合序列设计方案，见图 2.11 左图所示。它采用 3 个级数分别为 r、s、t 的线性移位寄存器组合而成，r、s、t 互素。请注意，寄存器 t 的输出须求反后再与寄存器 s 的输出相乘。该组合网路产生的非线性复合序列的周期为

$$T=(2^r-1)(2^s-1)(2^t-1) \tag{2-19}$$

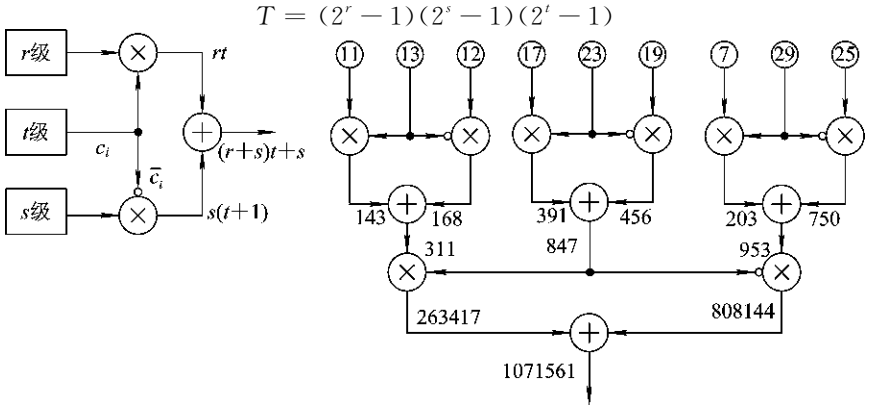

图 2.11　一种多路 m 序列非线性组合的实例

复杂度为

$$C(M) = (r+s)t + s \tag{2-20}$$

由多个这样的组合网路进一步组合，可产生复杂度很高的非线性组合序列，图 2.11 的右图是一个实例。

2.3.4 非线性复合序列

复合器实际上是一个地址选择器。由 A 寄存器的状态来选择 B 寄存器的某一位输出 $\{k_i\}$。例如图 2.12 所示的 $\mathrm{LFSR_A}$ 是三位，$\mathrm{LFSR_B}$ 是四位，复合器所设计的选择方式是由 $\mathrm{A_2A_1}$ 的码值决定 $\mathrm{LFSR_B}$ 的地址，正好选通 $\mathrm{LFSR_B}$ 四位中的任何一位。

如果 t_1 时刻 $\mathrm{A_2A_1A_0}=011$，$\mathrm{B_3B_2B_1B_0}=1100$，则选择器 $\mathrm{A_2A_1}=01$，选通了 $\mathrm{B_1}$ 的地址，于是 $\mathrm{B_1}=0$ 输出。如果 t_2 时刻 $\mathrm{A_2A_1A_0}=101$，$\mathrm{B_3B_2B_1B_0}=0110$，则 $\mathrm{A_2A_1}=10$，选通了 $\mathrm{B_2}=1$。

若 A、B 两个 LFSR 的级数 n 与 m 互素，则组合后输出序列的周期可达到

$$T = (2^n-1)(2^m-1) \tag{2-21}$$

而复杂度最大可达到

$$C(\{a\}) = n \times m \tag{2-22}$$

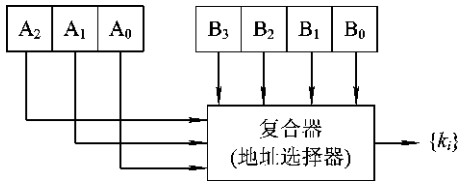

图 2.12 非线性复合序列发生器

2.4 利用线性反馈移位寄存器的密码反馈

反馈移位寄存器除了被用来产生密钥流之外，还可以直接把移位反馈原理用于密码的加密算法中。一般流程可示意为如图 2.13 所示。

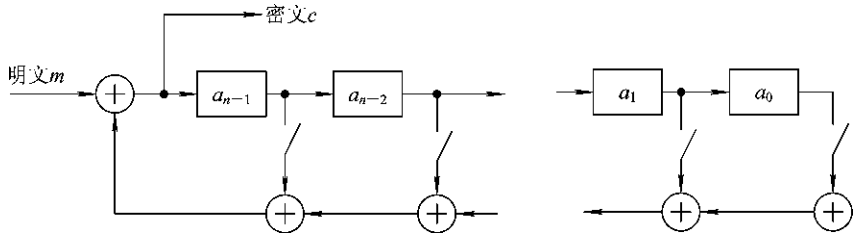

图 2.13 利用移位反馈原理的加密算法

解码流程只需把明文 m 与密文 c 的两处箭头方向相反即可。

将此原理推而广之，变通得到更多灵活的加密方法。如已知明文 m=“cryptography is the science of data security”，采用维吉利亚密码算法加密，密钥是 red star，则生成的密文是：

“TVBHMOXIESZRIJKLHKUIVEGHGYDRKEVWVUIZXB”

然而若采用密文反馈的方法，前七个字符用 redstar 为密钥，加密得到“TVBHMOX”后，相继的七个字符不用 redstar 为密钥，而用 TVBHMOX 为密钥来加密，又得到七字密文。再用此七字密文为密钥，来加密后继的七个明文字符，如此下去，得到的密文为

“TVBHMOXKVQOKWPDCUGMGTQEYURJTJEQYTDKRZO”

或者采用 redstar 与前七个字密文的模 26 加的结果作为密钥，来加密后继的七个明文字符，就可以克服只要知道了密钥长度，就能递推解密的漏洞。这样得到的密文是：

“TVBHMOXBZTGDWGLKAQYEBPQHWWHSZUCSRBAYRD”

习 题 2

1. 明文加密成密文有哪几种方式？
2. 什么是“一字一密”系统？
3. 什么是序列的复杂度？什么是算法的复杂度？
4. 将 LFSR 各级全部赋为 0，其输出流是什么？
5. 对于一个 48 位的 LFSR，有多少种可能的初始状态？
6. 在一个最大长度为 4 位的 LFSR 中，以 4 位和 3 位为反馈。对于从 0000～1111 的每一个可能的初值，其输出序列的周期为多少？
7. 已知序列密码的密文串 1010110110 和相应的明文串 0100010001，而且还已知密钥流是使用三级线性反馈移位寄存器产生的，试求出 LFSR 的反馈系数和初值。

实践练习 2

实验题目：m 序列为密钥的序列加密的密文破译。

实验平台：Mathematica 4.0。

实验内容：给出 m 序列为密钥的序列加密的密文和起始的一部分明文，进行破译训练。

实验步骤：

(1) 用已知的部分明文与密文模 2 加得到部分密钥。

(2) 由部分密钥构造不同维数的错位方阵，计算并寻找能使其行列式在模 2 运算下等于 1 的最大的维数，这就是产生该序列的 LFSR 的级数 n。

(3) 由已确定的 n 级错位方阵的逆矩阵乘以下一个状态，得到特征多项式。

(4) 由特征多项式构造 m 序列生成器。

(5) 由完整的 m 序列解密全部密文。

第3章 分组密码

分组密码是将明文分成固定长度的一些段落(分组),在密钥作用下逐段进行加密的方法。这样做的好处是处理速度快,可靠性高,软(硬)件都能实现,而且节省资源,容易标准化。因此,分组密码得到了广泛的应用,同时也使分组密码成为许多密码组件的基础,比如 MAC(消息认证码)系统。

3.1 DES

美国国家标准局 1977 年公布了由 IBM 公司研制的 Data Encryption Standard(DES)作为非机要部门的数据加密标准。它是迄今为止流行最广、时间最长的加密算法,也是现代分组加密技术的典型。原规定使用期 10 年,然而 10 年来并未发现有任何攻击能够威胁到它的安全,且比它更好的标准尚未产生,所以直到 20 世纪 90 年代,它一直在延期使用。可见它是很成功的。此后产生的许多加密方法都直接或间接地受到了它的启发。

3.1.1 DES 加密算法[2]

明文分组长 64 bit,$m=m_1, m_2, \cdots, m_{64}$。密钥长 56 bit,加上每 7 bit 一个奇偶校验位,共 64 bit。

加密过程可表达为

$$\mathrm{DES}(m) = \mathrm{IP}^{-1} \cdot T_{16} \cdot T_{15} \cdots T_2 \cdot T_1 \cdot \mathrm{IP}(m) \tag{3-1}$$

1. 置换与逆置换

IP 是初始置换,IP^{-1}是逆置换,分别按表 3.1 的序号置换数据的 bit 值。

表 3.1 DES 加密系统中的 IP 置换与逆置换表

IP								IP^{-1}							
58	50	42	34	26	18	10	2	40	8	48	16	56	24	64	32
60	52	44	36	28	20	12	4	39	7	47	15	55	23	63	31
62	54	46	38	30	22	14	6	38	6	46	14	54	22	62	30
64	56	48	40	32	24	16	8	37	5	45	13	53	21	61	29
57	49	41	33	25	17	9	1	36	4	44	12	52	20	60	28
59	51	43	35	27	19	11	3	35	3	43	11	51	19	59	27
61	53	45	37	29	21	13	5	34	2	42	10	50	18	58	26
63	55	47	39	31	23	15	7	33	1	41	9	49	17	57	25

不难验证：$IP * IP^{-1} = IP^{-1} * IP = 1$。

2. 迭代加密运算

将 IP **置换后的** 64 bit **明文分成两半，各** 32 bit，**分别进入加密器的左、右两个入口，先后经** T_1，T_2，…，T_{16}**进行** 16 **轮迭代加密运算。**

每轮加密 T_i**流程如图** 3.1 **所示。其中，**⊕**表示按位模** 2 **加；**f **表示扩展与收缩函数** $f(R_{i-1}, k_i)$**。处理后：**

$$L_i = R_{i-1}, R_i = L_{i-1} \oplus f(R_{i-1}, k_i) \tag{3-2}$$

值得注意的是，在最后一轮加密后，左、右两半不再交换位置。

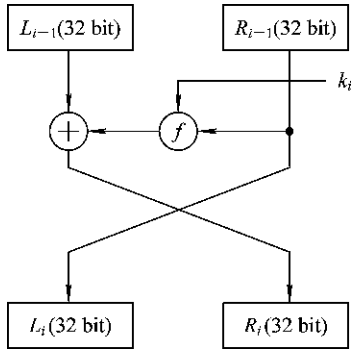

图 3.1 **第** i **轮加密** T_i **流程图**

3. 扩展与收缩函数

(R_{i-1}, k_i)**函数的具体结构如图** 3.2 **所示。**

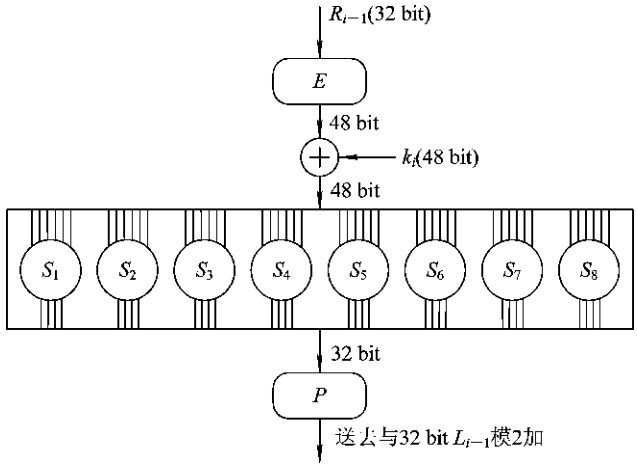

图 3.2 **扩展与收缩函数**(R_{i-1}, k_i)**的结构**

图 3.2 **中符号的说明如下：**

(1) E **为扩展变换，将** 32 bit **扩展为** 48 bit，**其方法是将信息的某些** bit **位重复：**

2 3 4 5 4 5 6 7 8 9 8 9 10 11 12 13 12 13 14 15 16 17 16 17 18 19 20 21 20 21

22 23 24 25 24 25 26 27 28 29 28 29 30 31 32 1 32 1

(2) ⊕**表示** 48 bit **明文与** 48 bit **密钥模** 2 **加。**

(3) **经** S **盒处理，将** 48 bit **数据变回** 32 bit。

48 bit 数据被分成 8 组，每组 6 bit，第 i 组为 $b_1b_2b_3b_4b_5b_6$，送入 S_i 处理。S_i 是一个 4 行 16 列的表，6 bit 输入数据中，b_1b_6 构成的二进制数给出行序号(0，1，2，3)，$b_2b_3b_4b_5$ 构成的二进制数给出列序号(0～15)。查表得到 0～15 的十进制数，化为二进制就是 4 bit 的输出数据($y_0y_1y_2y_3$)。8 个 S_i 分表共输出 32 bit。S 盒的结构如表 3.2 所示。

表 3.2　DES 加密系统中的 S 盒数据对照表

行＼列		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	10	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	11	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	00	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	01	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	10	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	11	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	00	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	01	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	10	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	11	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	00	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	01	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	11	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	00	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	01	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	10	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	00	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	01	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	10	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	11	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	00	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	01	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	10	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	11	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	00	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	01	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	10	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	11	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

(4) 再经置换 P，结束本轮加密，最终结果如表 3.3 所示。

表 3.3 $f(R_{i-1}, k_i)$函数中 P 置换的重排列次序

	y_0	y_1	y_2	y_3			y_0	y_1	y_2	y_3
S_1	1	2	3	4		S_1	16	7	20	21
S_2	5	6	7	8		S_2	29	12	28	17
S_3	9	10	11	12		S_3	1	15	23	26
S_4	13	14	15	16	P →	S_4	5	18	31	10
S_5	17	18	19	20		S_5	2	8	24	14
S_6	21	22	23	24		S_6	32	27	3	9
S_7	25	26	27	28		S_7	19	13	30	6
S_8	29	30	31	32		S_8	22	11	4	25

4. 子密钥的产生

从原始的密钥出发，为 16 轮加密产生出 16 个不同的子密钥 $k_i(i=1, 2, 3, \cdots, 16)$：

(1) 除去校验位(8、16、24、32、40、48、56、64 位)，并按 $PC-1$ 重排，如表 3.4 所示。

表 3.4 产生各轮子密钥时的 $PC-1$ 重排方式

C_0(左 28 bit)							D_0(右 28 bit)						
57	49	41	33	25	17	9	63	55	47	39	31	23	15
1	58	50	42	34	26	18	7	62	54	46	38	30	22
10	2	59	51	43	35	27	14	6	61	53	45	37	29
19	11	3	60	52	44	36	21	13	5	28	20	12	4

(2) C_0，D_0 是左右各半，分别按图 3.3 所示的流程进行处理。

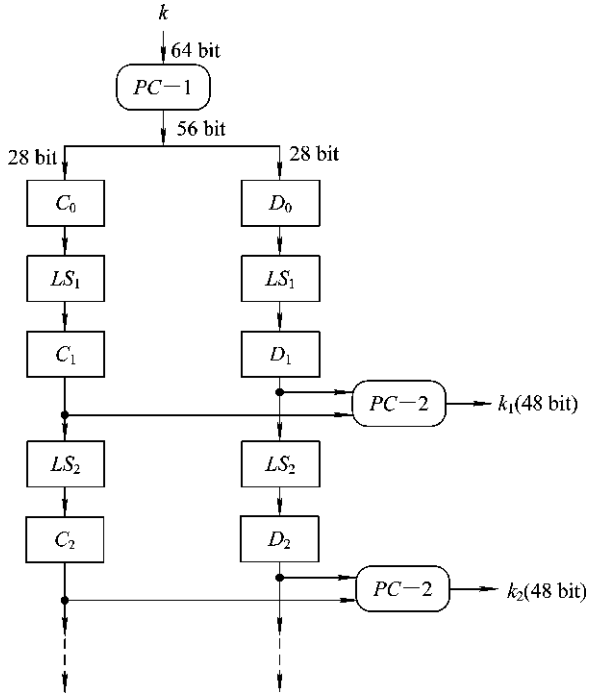

图 3.3 产生子密钥的流程图(只画出两个，其余相同)

(3) LS_j 是循环左移操作，移动位数因子密钥序号 j 而不同，由表 3.5 给出。

表 3.5 产生各轮子密钥时循环左移的位数

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
左移位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

(4) $PC-2$ 把左右两路数据合并，同时再次被重排次序，并且从 56 bit 中选出 48 bit 位(取掉了 9、18、22、25、35、38、43、54 位)，作为子密钥输出，如表 3.6 所示。

表 3.6 输出 48 bit 子密钥的各 bit 重排列次序

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

【例 1】 用密钥 program 对明文 computer 加密。

解：密钥和明文的 ASCII 码为

k=01110000 01110010 0110111 01100111 01110010 01100001 01101101(共 56 bit)

m=01100011 01101111 01101101 01110000 01110101 01110100 01100101 01110010

明文经 IP 置换后得：

L_0=11111111 10111000 01110110 01010111

R_0= 0000000 11111111 00000110 10000011

密钥经 $PC-1$ 分组置换后得：

C_0=11101100 10011001 00011011 1011

D_0=10110100 01011000 10001110 0110

各左移 1 位再通过 $PC-2$ 变换得 48 bit 子密钥：

k_1=00111101 10001111 11001101 00110111 00111111 01001000

R_0(32 bit)经 E 作用扩展为 48 bit：

R_0'=10000000 00010111 11111110 10000000 11010100 00000110

再与 k_1 相异或得：

$R_0' \oplus k_1$=10111101 10011000 00110011 10110111 11101011 01001110

分成 8 组：

101111，011001，100000，110011，101101，111110，101101，001110

通过 S 盒后输出 32 bit：

01110110 00110100 00100110 10100001

再经过 P 置换，这才得到(R_0，k_1)的结果：

01110110 00110100 00100110 10100001

然后与 L_0 模 2 加，赋值给 R_1，同时原来的 R_0 赋值给 L_1，完成第 1 轮加密。得到：

L_1=00000000 11111111 00000110 10000011

R_1=10111011 10011000 11101000 11001000

如此循环加密 16 次，得到：

L_{16}=0101000 10101000 01000001 10111000

R_{16}=01101001 11111110 10101110 00110011

DES的加密结果，每一比特都是明文m与密钥k的每一比特的复杂函数，明文或密钥每改变一个比特，都会对密文产生巨大影响。

3.1.2 解密算法

DES的解密十分简单，仍然使用加密一样的模块，次序倒过来就行了。这是因为：

$$\mathrm{DES}(m) = \mathrm{IP}^{-1} \cdot T_{16} \cdot T_{15} \cdots T_2 \cdot T_1 \cdot \mathrm{IP}(m)$$

若取

$$\mathrm{DES}^{-1}(c) = \mathrm{IP}^{-1} \cdot T_1 \cdot T_2 \cdots T_{15} \cdot T_{16} \cdot \mathrm{IP}(c) \tag{3-3}$$

就有

$$\begin{aligned}\mathrm{DES}^{-1}(c) &= \mathrm{DES}^{-1}[\mathrm{DES}(m)] \\ &= \mathrm{IP}^{-1} \cdot T_1 \cdot T_2 \cdots T_{15} \cdot T_{16} \cdot \mathrm{IP}\left[\ \mathrm{IP}^{-1} \cdot T_{16} \cdot T_{15} \cdots T_2 \cdot T_1 \cdot \mathrm{IP}(m)\ \right]\end{aligned}$$

首先，$\mathrm{IP} \cdot \mathrm{IP}^{-1}=1$，中间$\mathrm{IP} \cdot \mathrm{IP}^{-1}$相抵消后，两个$T_{16}$相连。加密过程的$T_{16}$处理前是$L_{15}R_{15}$，处理后

$$R_{16} = L_{15} \oplus f(R_{15}, k_{16}),\ L_{16} = R_{15}$$

再经解密过程的T_{16}处理(见图3.4)，得

$L = R_{15}$

$R = L_{15} \oplus f(R_{15}, k_{16}) \oplus f(R_{15}, k_{16}) = L_{15}$(模2加时，两个相同的处理必抵消)

两个T_{16}处理前是$L_{15}R_{15}$，处理后仍得到$L_{15}R_{15}$，可见$T_{16}T_{16}=1$，同理，各个

$$T_i T_i = 1 \quad i = 1, 2, 3, \cdots, 16$$

最后又是$\mathrm{IP} \cdot \mathrm{IP}^{-1}=1$，于是

$$\mathrm{DES}^{-1}(\mathrm{DES}(m)) = m$$

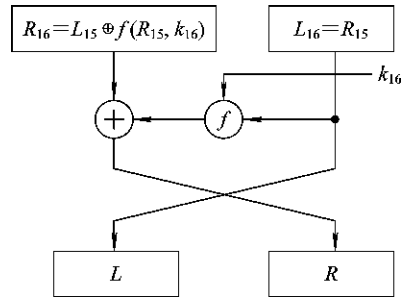

图3.4 $T_{16}T_{16}=1$的证明

3.1.3 关于DES的安全问题

1. 弱密钥

有个别密钥不适合于DES算法，比如使子密钥产生器中C_0和D_0为全0或全1的密钥，无论怎样循环移位都不变，致使16次迭代所用的子密钥不变，造成

$$T_1 = T_2 = \cdots = T_{16},\ \mathrm{DES}_k(m) = \mathrm{DES}_k^{-1}(m)$$

或写为

$$\mathrm{DES}_k(\mathrm{DES}_k(m)) = m$$

安全性就失去了保证。

这样的密钥最好不用，它们叫做弱密钥，共 4 个，用十六进制表示，它们是：

01 01 01 01 01 01 01 01，1F 1F 1F 1F 1F 1F 1F 1F，

E0 E0 E0 E0 E0 E0 E0 E0，FE FE FE FE FE FE FE FE，

还有 12 个半弱密钥 k 和 k'，它们成对使

$$\mathrm{DES}_k(m)=\mathrm{DES}_{k'}^{-1}(m)$$

它们是：

01 FE 01 FE 01 FE 01 FE 和 FE 01 FE 01 FE 01 FE 01，

1F E0 1F E0 0E F1 0E F1 和 E0 1F E0 1F F1 0E F1 0E，

01 E0 01 E0 01 F1 01 F1 和 E0 01 E0 01 F1 01 F1 01，

1F FE 1F FE 0E FE 0E FE 和 FE 1F FE 1F FE 0E FE 0E，

01 1F 01 1F 01 0E 01 0E 和 1F 01 1F 01 0E 01 0E 01，

E0 FE E0 FE F1 FE F1 FE 和 FE E0 FE E0 FE F1 FE F1

2. DES 的安全性

DES 由于未遇到敌手而超期服役，直到 20 世纪 90 年代，Shamir 等人提出“差分分析法”，才对 DES 构成了理论上的威胁。后来的“线性逼近法”需要已知明文，且需要 $2^{43}=4.398\times10^{12}$ 对明密文，联合十多台工作站工作十多天才可以搜索到密钥。

DES 终于完成了它的历史使命，但它的思想还是值得借鉴的。分析表明，DES 的薄弱之处不是算法，而是密钥太短，只有 56 bit，7 个英文字符！为遍历法搜索提供了可能。

3.1.4 DES 的变形(改进)

1. 三重加密方式

针对 DES 密钥太短的问题，提出了如图 3.5 所示的三重加密方式，使它的复杂度增加。

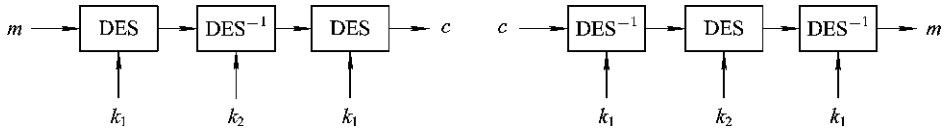

图 3.5 两种三重 DES 加密方式

2. 密文分组连接方式(CBC)

原来的做法是将明文分组加密后，把各组密文连接起来，称为电码本方式(ECB)。现改为第一组加密的结果，与第二组明文相加后再加密。一方面将此密文组输出，另一方面与下一组明文相加后再加密，作为下一密文分组，最后将各组加密结果链接，如图 3.6(a)所示。

3. 密文反馈方式(CFB)

如图 3.6(b)所示，每块(分组)与前块送过来的加密结果相异或后，作为本块输出，并加密后送往下一级作同样的处理。

4. 输出反馈方式(OFB)

如图 3.6(c)所示，初始矢量被一次次地加密，分别与每块(分组)数据相异或后，作为本块输出，这样，各块的密文就是相互独立的，不再相互影响。

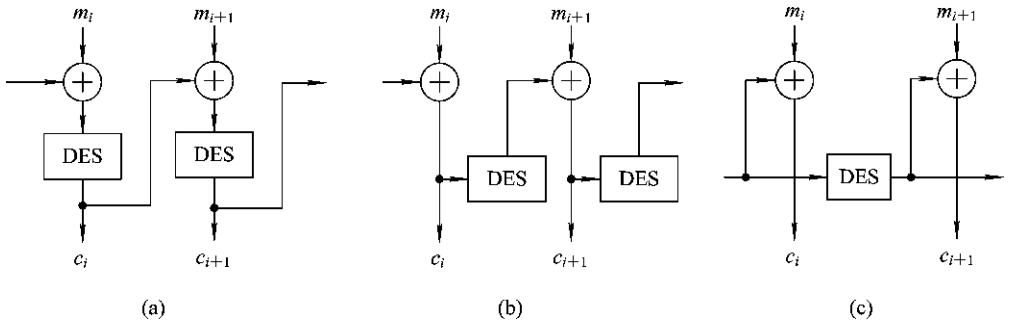

图 3.6 三种改进的 DES 加密方式

(a) CBC 方式；(b) CFB 方式；(c) OFB 方式

3.2 IDEA

20 世纪 90 年代，出现了很多优秀的加密算法，本节介绍其中三种。

3.2.1 IDEA 算法[2]

IDEA(International Data Encryption Algorithm，国际数据加密算法)是由中国学者朱学嘉博士和 James Massey 在 1990 年合作提出的，1992 年完成。它的加解密运算速度都很快，无论是用软件实现，还是用硬件实现都不难，成为替代 DES 的优选算法之一。

IDEA 的密钥是 128 bit，而明文分组仍为 64 bit，分成四个子块 $X_1X_2X_3X_4$，各 16 bit，进行 8 轮循环迭代加密运算。每次加密的过程如图 3.7 所示。

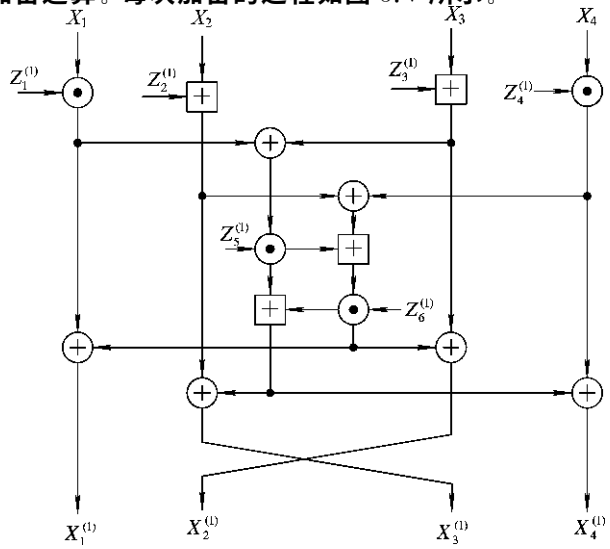

图 3.7 IDEA 迭代加密运算原理

图 3.7 中，⊕表示模 2 加(异或)；⊙表示模 $2^{16}+1$ 的乘法；⊞表示模 2^{16} 的加法。$Z_1^{(1)}\sim Z_6^{(1)}$ 是第一轮加密使用的六个子密钥，它们来自 128 bit 密钥的顺序分组(每组 16 bit，共 8 组)的前六组。第二轮处理的算法相同，只是所用的子密钥不同，$Z_1^{(2)}$ 和 $Z_2^{(2)}$ 是第一轮子密钥取剩下的两个分组，$Z_3^{(2)}\sim Z_6^{(2)}$ 则来自 128 bit 密钥循环左移 25 位后再分成 8 组的前四个分组，而后四个分组留给第三轮子密钥的 $Z_1^{(3)}\sim Z_4^{(3)}$，$Z_5^{(3)}$ 和 $Z_6^{(3)}$ 则来自 128 bit 钥再次循环左移 25 位之后的 8 个分组。如此下去，直到第八轮迭代。第九轮不同于前八轮，不再需要 $Z_5^{(9)}$ 和 $Z_6^{(9)}$ 有关的迭代加密过程，实际上只有如图 3.8 所示的一步。最后将 $X_1^{(9)}\sim X_4^{(9)}$ 连接起来即是密文。

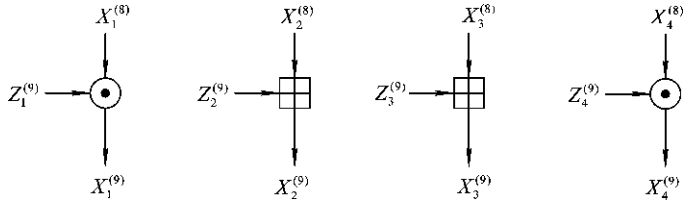

图 3.8　IDEA 迭代加密的第九轮运算

IDEA 解密和加密算法相同，只是子密钥不同。加、解密的密钥如下：

轮次	加密子密钥	解密子密钥
1	$Z_1^{(1)}, Z_2^{(1)}, Z_3^{(1)}, Z_4^{(1)}, Z_5^{(1)}, Z_6^{(1)}$	$(Z_1^{(9)})^{-1}, -Z_2^{(9)}, -Z_3^{(9)}, (Z_4^{(9)})^{-1}, Z_5^{(8)}, Z_6^{(8)}$
2	$Z_1^{(2)}, Z_2^{(2)}, Z_3^{(2)}, Z_4^{(2)}, Z_5^{(2)}, Z_6^{(2)}$	$(Z_1^{(8)})^{-1}, -Z_2^{(8)}, -Z_3^{(8)}, (Z_4^{(8)})^{-1}, Z_5^{(7)}, Z_6^{(7)}$
3	$Z_1^{(3)}, Z_2^{(3)}, Z_3^{(3)}, Z_4^{(3)}, Z_5^{(3)}, Z_6^{(3)}$	$(Z_1^{(7)})^{-1}, -Z_2^{(7)}, -Z_3^{(7)}, (Z_4^{(7)})^{-1}, Z_5^{(6)}, Z_6^{(6)}$
4	$Z_1^{(4)}, Z_2^{(4)}, Z_3^{(4)}, Z_4^{(4)}, Z_5^{(4)}, Z_6^{(4)}$	$(Z_1^{(6)})^{-1}, -Z_2^{(6)}, -Z_3^{(6)}, (Z_4^{(6)})^{-1}, Z_5^{(5)}, Z_6^{(5)}$
5	$Z_1^{(5)}, Z_2^{(5)}, Z_3^{(5)}, Z_4^{(5)}, Z_5^{(5)}, Z_6^{(5)}$	$(Z_1^{(5)})^{-1}, -Z_2^{(5)}, -Z_3^{(5)}, (Z_4^{(5)})^{-1}, Z_5^{(4)}, Z_6^{(4)}$
6	$Z_1^{(6)}, Z_2^{(6)}, Z_3^{(6)}, Z_4^{(6)}, Z_5^{(6)}, Z_6^{(6)}$	$(Z_1^{(4)})^{-1}, -Z_2^{(4)}, -Z_3^{(4)}, (Z_4^{(4)})^{-1}, Z_5^{(3)}, Z_6^{(3)}$
7	$Z_1^{(7)}, Z_2^{(7)}, Z_3^{(7)}, Z_4^{(7)}, Z_5^{(7)}, Z_6^{(7)}$	$(Z_1^{(3)})^{-1}, -Z_2^{(3)}, -Z_3^{(3)}, (Z_4^{(3)})^{-1}, Z_5^{(2)}, Z_6^{(2)}$
8	$Z_1^{(8)}, Z_2^{(8)}, Z_3^{(8)}, Z_4^{(8)}, Z_5^{(8)}, Z_6^{(8)}$	$(Z_1^{(2)})^{-1}, -Z_2^{(2)}, -Z_3^{(2)}, (Z_4^{(2)})^{-1}, Z_5^{(1)}, Z_6^{(1)}$
9	$Z_1^{(9)}, Z_2^{(9)}, Z_3^{(9)}, Z_4^{(9)}$	$(Z_1^{(1)})^{-1}, -Z_2^{(1)}, -Z_3^{(1)}, (Z_4^{(1)})^{-1}$

其中，Z^{-1} 是 Z 的模$(2^{16}+1)$的乘法逆元，$-Z$ 是 Z 的模 2^{16} 的加法逆元。

由于 IDEA 的密钥长度是 DES 的一倍，所以用同样的方式攻击 IDEA，所需工作量是 DES 的 $2^{72}=4.7\times10^{21}$ 倍。许多科研部门和军事部门对 IDEA 进行攻击测试，未见成功报道。

【例 2】　密钥为 computer security，对明文 Tsinghua 加、解密。

解：密钥和明文的 ASCII 码为

k=11000110 11110110 10110110 00001110 …(共 8×16=128 bit)

m=00101010 11001110 10010110 01110110 …(共 8×8=64 bit)

$X_1=(0111001101010100)_2=29524$（注意先输入的为低位，后输入的为高位）

$Z_1^{(1)}=(0110111101100011)_2=28515$（注意高低位顺序）

$X_1\odot Z_1^{(1)}=X_1\cdot Z_1^{(1)} \bmod (2^{16}+1)=29524\times28515 \bmod 65537=54091$

$=(1101001101001111)_2$

$X_1\boxplus Z_1^{(1)}=(0110111001101001)_2\boxplus(0111000001101101)_2=28265+28781 \bmod 65536$

$=57046=(1101111011010110)_2$

按算法迭代 8 次后，得密文：

c=11011000 00110011 01101000 11010010 …（共 8×8=64 bit）

当密钥 k 改变一位时，所得的密文有 29 位改变；当明文 m 改变一位时，所得的密文有 31 位改变。

3.2.2 NSSU

NSSU 是前苏联国家标准，密钥为 256 bit，迭代 32 次。明文分成左右 32 bit，第 i 轮加密钥：

$$L_i = R_{i-1},\ R_i = L_{i-1} \oplus (R_{i-1},\ k_i) \tag{3-4}$$

k_i 是第 i 轮的子密钥。第 i 轮加密流程如图 3.9 所示。

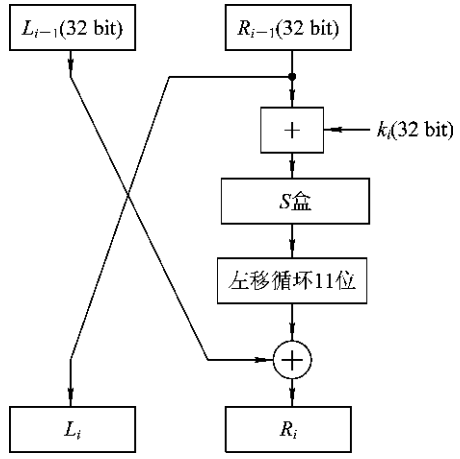

图 3.9 NSSU 的第 i 次加密流程

函数 $f(R_{i-1},\ k_i)$的定义是首先 R_{i-1}与 k_i 作模 2^{32}的加法，即

$$S \leftarrow (R_{i-1} \oplus k_i) \bmod 2^{32}$$

然后将 S 分成 8 组，每组 4 bit，输入到 8 个 S 盒中。S 盒的结构见表 3.7。

S 盒的输入$(i_1,\ i_2,\ i_3,\ i_4)$ 与输出$(j_1,\ j_2,\ j_3,\ j_4)$ 的关系为

$$(j_1,\ j_2,\ j_3,\ j_4) = S_k(i_1,\ i_2,\ i_3,\ i_4)$$

表 3.7 NSSU 的 S 盒结构

序号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	4	10	9	2	13	8	0	14	6	11	1	12	7	15	5	3
S_2	14	11	4	12	6	13	15	10	2	3	8	1	0	7	5	9
S_3	5	8	1	13	10	3	4	2	14	15	12	7	6	0	9	1
S_4	7	13	10	1	0	8	9	15	14	4	6	12	11	2	5	3
S_5	6	12	7	1	5	15	13	8	4	10	9	14	0	3	11	2
S_6	4	11	10	0	7	2	1	13	3	6	8	5	9	12	15	14
S_7	13	11	4	1	3	15	5	9	0	10	14	7	6	8	2	12
S_8	1	15	13	0	5	7	10	4	9	2	3	14	6	11	8	12

例如，第 1 分组

$$(i_1, i_2, i_3, i_4) = (1001)_2 = 9$$

查 S 盒的第 1 行，有

$$S_1(9) = 11 = (1011)_2$$

则 $$(j_1, j_2, j_3, j_4) = 1011$$

即

$$j_1 = 1, j_2 = 0, j_3 = 1, j_4 = 1$$

又如，第 7 分组

$$(i_1, i_2, i_3, i_4) = (1101)_2 = 13$$

查 S 盒的第 7 行，有

$$S_1(13) = 8 = (1000)_2$$

则输出为

$$j_1 = 1, j_2 = 0, j_3 = 0, j_4 = 0$$

S 盒的全体输出为 32 bit，循环左移 11 位后再与 L_{i-1} 作⊕运算，即得 R_i。

子密钥的产生异常简单：将 256 bit 密钥分成 8 份，每份 32 bit，称为一个子密钥。每轮所用的密钥按表 3.8 从这 8 个子密钥中选取。

表 3.8 NSSU 的子密钥选取

轮次	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
子密钥	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
轮次	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
子密钥	1	2	3	4	5	6	7	8	8	7	6	5	4	3	2	1

3.2.3 TEA

TEA 由英国剑桥大学的 David. W 和 Roger. N 提出，特点是加密速度极快，抗差分攻击能力强。

TEA 的明文为 64 bit，密钥为 128 bit，算法十分简单。它的迭代次数可变，32 次很充分，16 次足够，8 次亦可行。算法如下：

S1(初始化)

明文分成 $v(0)$ 和 $v(1)$ 两部分，各 32 bit，赋值：

$y \leftarrow v(0)$，$z \leftarrow v(1)$，Sum $\leftarrow 0$，Delta $\leftarrow$ 9E3779B9(十六进制数)

密钥分成 $k(0)$，$k(1)$，$k(2)$，$k(3)$ 四部分，各 32 bit，且

$a \leftarrow k(0)$，$b \leftarrow k(1)$，$c \leftarrow k(2)$，$d \leftarrow k(3)$，$n \leftarrow 32$

S2 若 $n>0$，则转 S3，否则转 S4。

S3 Sum ←Sum + Delta；

$y \leftarrow y+(z<<4)+a \wedge z+\text{Sum} \wedge (z>>5)+b$；

$z \leftarrow z+(y<<4)+c \wedge y+\text{Sum} \wedge (y>>5)+d$；

$n \leftarrow n-1$，转 S2。

S4 $v(0)\leftarrow y$，$v(1)\leftarrow z$，结束。

其中，$\wedge$是按位异或，即模2加⊕；$<<$是左移，高位舍弃，低位补零；$>>$是右移，低位舍弃，高位补零。

TEA的解密算法如下：

S1 （初始化）

$y\leftarrow v(0)$，$z\leftarrow v(1)$，Delta$\leftarrow$9E3779B9，Sum$\leftarrow$C6EF3720；

$a\leftarrow k(0)$，$b\leftarrow k(1)$，$c\leftarrow k(2)$，$d\leftarrow k(3)$，$n\leftarrow 32$

S2 若$n>0$，则转S3，否则转S4。

S3 $z\leftarrow z-(y<<4)+c\wedge y+\text{Sum}\wedge(y>>5)+d$；

$y\leftarrow y-(z<<4)+a\wedge z+\text{Sum}\wedge(z>>5)+b$；

Sum$\leftarrow$Sum－Delta；

$n\leftarrow n-1$，转S2。

S4 $v(0)\leftarrow y$，$v(1)\leftarrow z$，结束。

3.3 *AES*

AES是21世纪分组加密技术的最新发展。1999年美国政府向全世界公开征求下一代密码算法，以取代DES标准。这次活动是1997年4月15日由美国标准技术研究所(NIST)发起的，称为AES(Advanced Encxyption Standard)活动。1999年从初选的15个算法中筛选了5个为候选标准，它们是MARS、RC6、Rijndael、Serpent和Twofish。2000年10月2日正式宣布Rijndael将不加修改地作为AES标准算法。

3.3.1 Rijndael算法[4]

Rijndael算法是比利时学者J. Daemaen和V. Rijndael提出的，它采用代替置换网络，即SP结构进行迭代运算。每一轮由三层组成：线性混合层，确保多轮之上的高度扩散；非线性层，由非线性S盒构成，起到混淆的作用；密钥加层，把子密钥简单异或到中间状态上。S盒采用GF(2^8)域中的乘法逆运算，本原多项式取为$m(x)=x^8+x^4+x^3+x+1$，即十六进制的“11B”。它的差分均匀性和线性偏差均达到最佳。

Rijndael是一个数据块长度和密钥长度都可变的分组密码。数据块和密码长度可以分别是128 bit、192 bit、256 bit。首先把数据块写成“字”的形式，每个字包含4个字节，每个字节8 bit。密钥也写为字的形式(见下面的数据)，下列数据中每列是一个字。

$$a_{00}\ a_{01}\ a_{02}\cdots;\ k_{00}\ k_{01}\ k_{02}\cdots$$
$$a_{10}\ a_{11}\ a_{12}\cdots;\ k_{10}\ k_{11}\ k_{12}\cdots$$
$$a_{20}\ a_{21}\ a_{22}\cdots;\ k_{20}\ k_{21}\ k_{22}\cdots$$
$$a_{30}\ a_{31}\ a_{32}\cdots;\ k_{30}\ k_{31}\ k_{32}\cdots$$

设数据块字数为N_b，密钥字数为N_k；则迭代次数N_r的数值取决于N_b和N_k，其值由表3.9决定。

表 3.9 迭代次数 N_r 取决于 N_b 和 N_k

N_r	$N_b=4$	$N_b=6$	$N_b=8$
$N_k=4$	10	12	14
$N_k=6$	12	12	14
$N_k=8$	14	14	14

加密过程按图 3.10 算法迭代 N_r 次，最后一轮迭代没有列混合，其他各轮相同。

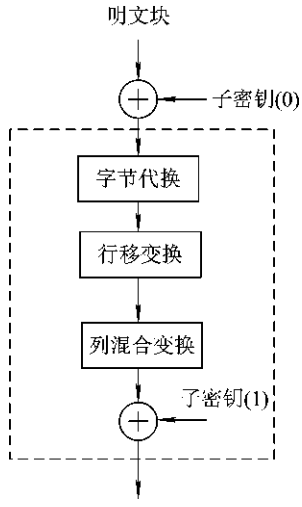

图 3.10 Rijndael 算法的一次迭代流程

1. 字节变换(ByteSub)

$\boldsymbol{a}_j$ 是第 j 列的字：

$$\text{ByteSub}(\boldsymbol{a}_j)=(\text{ByteSub}(a_{0j}),\ \text{ByteSub}(a_{1j}),\ \text{ByteSub}(a_{2j}),\ \text{ByteSub}(a_{3j})) \tag{3-5}$$

它的变换为

$$\text{ByteSub}(a_{ij})=\begin{pmatrix}10001111\\11000111\\11100011\\11110001\\11111000\\01111100\\00111110\\00011111\end{pmatrix} a_{ij}^{-1}+\begin{pmatrix}1\\1\\0\\0\\0\\1\\1\\0\end{pmatrix} \tag{3-6}$$

式中，a_{ij}^{-1} 是 a_{ij} 在 $\text{GF}(2^8)$ 域的乘法逆元。

实际上，字节变换可以事先计算出来，以列表形式存储，就如同 S 盒一样，靠查表就能方便地完成变换。这张表从 0→15 共 16 行、16 列(见表 3.10)。比如欲变换的字节是'8B'，则查'8'行'B'列，结果是'3D'。

表 3.10 Rijndael 的 S 盒

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	DB	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	F2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	3F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	45	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3R	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	BF	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

2. 行移变换(ShiftRow)

数据块第 0 行不变，第一行循环左移 c_1 字节，第二行循环左移 c_2 字节，第三行循环左移 c_3 字节。$c_1c_2c_3$ 的值根据 N_b 的大小确定，见表 3.11。

表 3.11 行移变换中的循环左移值

N_b	c_1	c_2	c_3
4	1	2	3
6	1	2	3
8	1	3	4

3. 列混合变换(MixColumn)

$$\text{MixColumn}(\boldsymbol{a}_j) = \boldsymbol{c} \otimes \boldsymbol{a}_j \tag{3-7}$$

其中，$\boldsymbol{a}_j$ 看成是 $\text{GF}(2^8)[x]/(x^4+1)$这个环中的元素；$\otimes$是环中的乘法，定义为

$$\text{MixColumn}\begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix} = \begin{pmatrix} c_0 & c_3 & c_2 & c_1 \\ c_1 & c_0 & c_3 & c_2 \\ c_2 & c_1 & c_0 & c_3 \\ c_3 & c_2 & c_1 & c_0 \end{pmatrix} \cdot \begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix} \tag{3-8}$$

$\boldsymbol{c}='03'x^3+'01'x^2+'01'x+'02'$是一个常矢量，于是列混合变换可以简单地当作一个矩阵相乘运算，不过相乘的各个矩阵元都是 $\text{GF}(2^8)$域的元素，遵循域运算规则而已。之所以选 $c_3='03'$，$c_2='01'$，$c_1='01'$，$c_0='02'$，为的是计算简单，同时由于 $\boldsymbol{c}$ 与 x^4+1 互素，

因此 c 有逆元：

$$\boldsymbol{d} = '0B'x^3 + '0D'x^2 + '09'x + '0E'$$

因而解密时的逆矩阵存在。

4. 子密钥的产生

N_r 轮加密共需 N_r+1 个子密钥，每个子密钥均由 N_k 个字（每字又包含 4 个字节）构成。比如 $N_b=4$ 且 $N_k=4$ 时，$N_r=10$，加密共需 11 个子密钥，第 i 个子密钥由 4 个字构成，它们是：

$$W(4i), W(4i+1), W(4i+2), W(4i+3) \qquad i = 0, 1, 2, \cdots, 10$$

第 0 个子密钥用于 10 轮加密之前，它就是原始密钥：

$$W(0) = (k_{00}, k_{10}, k_{20}, k_{30}), W(1) = (k_{01}, k_{11}, k_{21}, k_{31})$$
$$W(2) = (k_{02}, k_{12}, k_{22}, k_{32}), W(3) = (k_{03}, k_{13}, k_{23}, k_{33})$$

其他 10 个子密钥均由原始密钥的扩展与重组得到。其方法是：

(1) 如果 j 不是 4 的倍数，则 $W(j)=W(j-4)\oplus W(j-1)$，比如 $W(5)=W(1)\oplus W(4)$。

(2) 如果 j 是 4 的倍数，比如 $W(4)$、$W(8)$…… 则 $W(j)=W(j-4)\oplus T(W(j-1))$。其中，$T(W(j-1))$是对 $W(j-1)$作变换得到的：首先将 $W(j-1)$的 4 个字节作左移轮换，即$(k_0, k_1, k_2, k_3)\rightarrow(k_1, k_2, k_3, k_0)$，然后做字节变换，即通过查 S 盒对 4 个字节作替换，最后得到

$$T(W(j-1)) = (e \oplus r(j), f, g, h)$$

式中，(e, f, g, h)是 S 盒替换后的 4 个字节数据，而 $r(j)='01'\times 2^{(j-4)/4}$是 $GF(2^8)$域中计算的循环常数。即 $r(4)='01'$，其他 $j=4i$ 时，$r(j)=2r(j-4)$。于是 10 轮循环常数分别求出为$'01'$, $'02'$, $'04'$, $'08'$, $'10'$, $'20'$, $'40'$, $'80'$, $'1B'$, $'36'$（注意这里是域运算：$2\times'80'='100' \bmod m(x)='1B'$）。

在 N_b 和 N_k 为其他值的情况下，子密钥扩展的普遍公式是：

当 $i=0, 1, \cdots, N_k-1$ 时，有

$$\boldsymbol{W}_i = \boldsymbol{k}_i \tag{3-9}$$

当 $N_k \leqslant i \leqslant N_b(N_r+1)-1$ 时，有

$$W_i = \begin{cases} W_{i-N_k} \oplus \mathrm{ByteSub}(\mathrm{Rotate}(\boldsymbol{W}_{i-1})) \oplus R_{\mathrm{con}}\left[\dfrac{i}{N_k}\right] & \text{当 } i \text{ 是 } N_k \text{ 的整数倍时} \\ W_{i-N_k} \oplus \mathrm{ByteSub}\{\boldsymbol{W}_{i-1}\} & \text{当 } N_k > 6 \text{ 且 } i(\bmod N_k) = 4 \text{ 时} \\ W_{i-N_k} \oplus \boldsymbol{W}_{i-1} & \text{其他} \end{cases} \tag{3-10}$$

式，$\mathrm{Rotate}(a, b, c, d)$表示左移一个字节，即

$$\mathrm{Rotate}(a, b, c, d) = (b, c, d, a)$$

且

$$R_{\mathrm{con}}[i]=(RC[i], '00', '00', '00')\in GF(2^8)[x]/(x^4+1)$$

而

$$RC[i]=x^{i-1}\in GF(2^8)$$

最后，第 i 个子密钥选取为 $\boldsymbol{W}_{N_b\times i}$, $\boldsymbol{W}_{N_b\times i+1}$, …, $\boldsymbol{W}_{N_b(i+1)-1}$。

5. Rijndael 的安全性

Rijndael 的安全性表现在：

（1）对密钥没有任何限制，不存在弱密钥。

（2）能有效抵抗目前已知的各种攻击方法，如差分攻击、相关密钥攻击、插值攻击等。

（3）良好的理论基础使设计者可高强度地隐藏信息。

（4）关键常数的巧妙选择使计算速率可达 1 Gbit/s。

（5）可以实现 MAC、Hash、同步流密码、随机数生成等其他功能。

Rijndael 目前已经有效地应用在奔腾机、智能卡、ATM 机、B－ISDN、卫星通信等方面。

3.3.2 Camellia 算法[4]

Camellia 算法由日本三菱公司和日本电话电报公司联合设计，支持 128 bit 分组的明文和 128 bit、196 bit、256 bit 的密钥，达到了 AES 的要求。

1. Camellia 的构造

1）加密过程

对于 128 bit 的密钥，规定迭代 18 轮，每轮加密都相同，如图 3.11 所示；对于 192 bit 和 256 bit 的密钥，应迭代 24 轮，只需在图 3.12 中再增加 6 轮加密和一对 FL 和 FL^{-1} 处理即可。

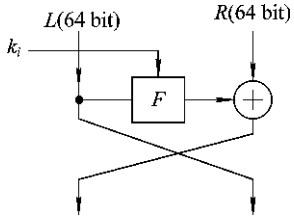

图 3.11 Camellia 算法的某一轮加密

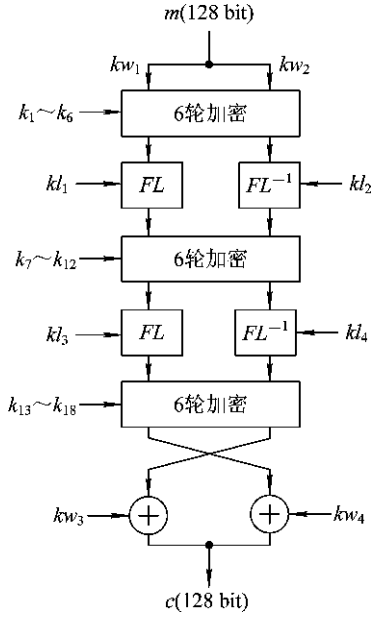

图 3.12 Camellia 算法的 18 轮加密

2）解密过程

与加密过程类似，只是颠倒密钥顺序。

3）密钥方案

首先根据三种不同长度的密钥 K，用不同方式给 128 bit 的 K_L 和 K_R 赋值，再由图

3.13 的方式生成两个 128 bit 的中间变量 K_A 和 K_B。

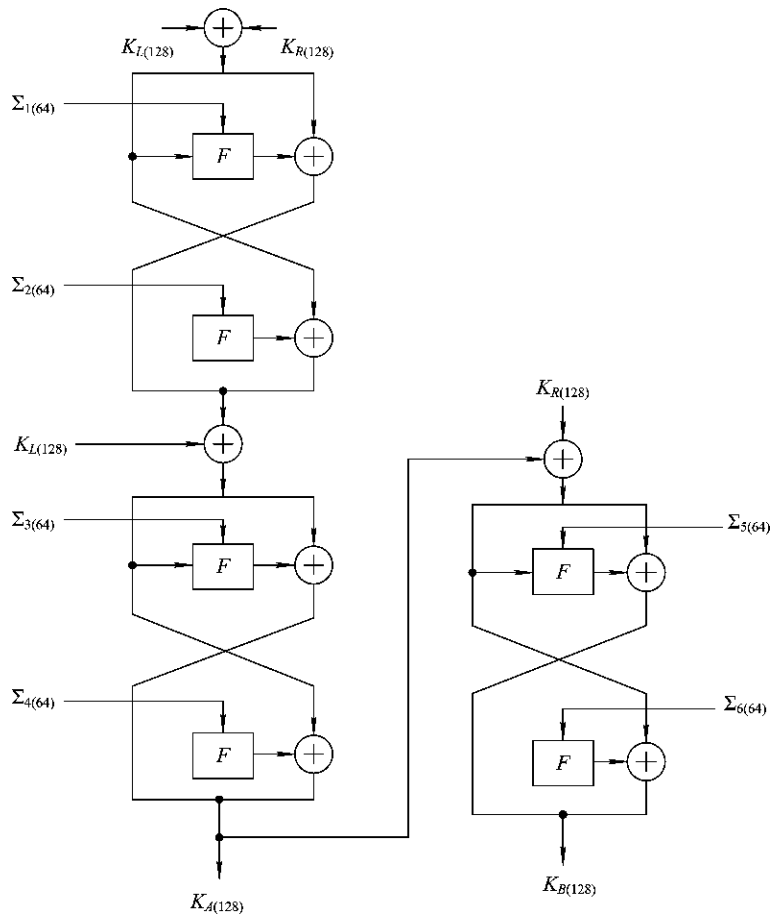

图 3.13 Camellia 算法的密钥方案

对 128 bit 的密钥 K，有 $K_L=K$，$K_R=0$（128 个 0）；对 192 bit 的密钥 K，有 $K_L=K$（前 128 bit），$K_R=K$（后 64 bit）$\|\ \overline{K(\text{后 64 bit})}$；对 256 bit 的密钥 K，有 $K_L=K$（前 128 bit），$K_R=K$（后 128 bit）。其中，K（前 128 bit）表示 K 的前面 128 bit；K（后 64 bit）表示 K 的后面 64 位；$\overline{K(\text{后 64 bit})}$ 表示 K 的后面 64 位的反码；$\|$ 表示顺序连接。

图 3.13 中所用的常数 $\sum_i$ 定义为第 i 个素数的平方根的十六进制表达的第二位到第十七位连续值。它们是：

$$\sum_1=\text{A09E667F3BCC908B},\quad \sum_2=\text{B67AE8584CAA73B2}$$
$$\sum_3=\text{C6EF372FE94F82BE},\quad \sum_4=\text{54FF53A5F1D36F1C}$$
$$\sum_5=\text{10E527FADE682D1D},\quad \sum_6=\text{B05688C2B3E6C1FD}$$

然后，分别由 $K_LK_RK_AK_B$ 循环移位生成加密所需要的各个子密钥。18（或 24）轮加密运算中共用到 18（或 24）个子密钥 k_i，4 个子密钥 kw_i，4 个子密钥 kl_i，它们均由 $K_LK_RK_AK_B$ 循环移位生成，循环方式见表 3.12 和表 3.13。

表 3.12 128 bit 密钥生成的子密钥

kw_1 $(K_L<<<0)L_{(64)}$	k_1 $(K_A<<<0)L_{(64)}$	k_{10} $(K_L<<<60)R_{(64)}$
kw_2 $(K_L<<<0)R_{(64)}$	k_2 $(K_A<<<0)R_{(64)}$	k_{11} $(K_A<<<60)L_{(64)}$
kw_3 $(K_A<<<111)L_{(64)}$	k_3 $(K_L<<<15)L_{(64)}$	k_{12} $(K_A<<<60)R_{(64)}$
kw_4 $(K_A<<<111)R_{(64)}$	k_4 $(K_L<<<15)R_{(64)}$	k_{13} $(K_L<<<94)L_{(64)}$
	k_5 $(K_A<<<15)L_{(64)}$	k_{14} $(K_L<<<94)R_{(64)}$
kl_1 $(K_A<<<30)L_{(64)}$	k_6 $(K_A<<<15)R_{(64)}$	k_{15} $(K_A<<<94)L_{(64)}$
kl_2 $(K_A<<<30)R_{(64)}$	k_7 $(K_L<<<45)L_{(64)}$	k_{16} $(K_A<<<94)R_{(64)}$
kl_3 $(K_L<<<77)L_{(64)}$	k_8 $(K_L<<<45)R_{(64)}$	k_{17} $(K_L<<<111)L_{(64)}$
kl_4 $(K_L<<<77)R_{(64)}$	k_9 $(K_A<<<45)L_{(64)}$	k_{18} $(K_L<<<111)R_{(64)}$

表 3.13 192 bit 和 256 bit 密钥生成的子密钥

kw_1 $(K_L<<<0)L_{(64)}$	k_1 $(K_B<<<0)L_{(64)}$	k_9 $(K_L<<<45)L_{(64)}$	k_{17} $(K_L<<<77)L_{(64)}$
kw_2 $(K_L<<<0)R_{(64)}$	k_2 $(K_B<<<0)R_{(64)}$	k_{10} $(K_L<<<45)R_{(64)}$	k_{18} $(K_L<<<77)R_{(64)}$
kw_3 $(K_B<<<111)L_{(64)}$	k_3 $(K_R<<<15)L_{(64)}$	k_{11} $(K_A<<<45)L_{(64)}$	k_{19} $(K_R<<<94)L_{(64)}$
kw_4 $(K_B<<<111)R_{(64)}$	k_4 $(K_R<<<15)R_{(64)}$	k_{12} $(K_A<<<45)R_{(64)}$	k_{20} $(K_R<<<94)L_{(64)}$
kl_1 $(K_R<<<30)L_{(64)}$	k_5 $(K_A<<<15)L_{(64)}$	k_{13} $(K_R<<<60)L_{(64)}$	k_{21} $(K_A<<<94)R_{(64)}$
kl_2 $(K_R<<<30)R_{(64)}$	k_6 $(K_A<<<15)R_{(64)}$	k_{14} $(K_R<<<60)R_{(64)}$	k_{22} $(K_A<<<94)L_{(64)}$
kl_3 $(K_L<<<60)L_{(64)}$	k_7 $(K_B<<<30)L_{(64)}$	k_{15} $(K_B<<<60)L_{(64)}$	k_{23} $(K_L<<<111)L_{(64)}$
kl_4 $(K_L<<<60)R_{(64)}$	k_8 $(K_B<<<30)R_{(64)}$	k_{16} $(K_B<<<60)R_{(64)}$	k_{24} $(K_L<<<111)R_{(64)}$

2. Camellia 组件

1) F 函数

图 3.14 表示的 F 函数将 64 bit 数据用 64 bit 密钥加密，具体结构为

$$Z'_{(64)} = F(X_{(64)}, k_{i(64)}) = P(S(X_{(64)} \oplus k_{i(64)})) \tag{3-11}$$

式中，S 是 8 个并行的 8×8 的 S 盒代替运算；P 是基于字节的线性变换。

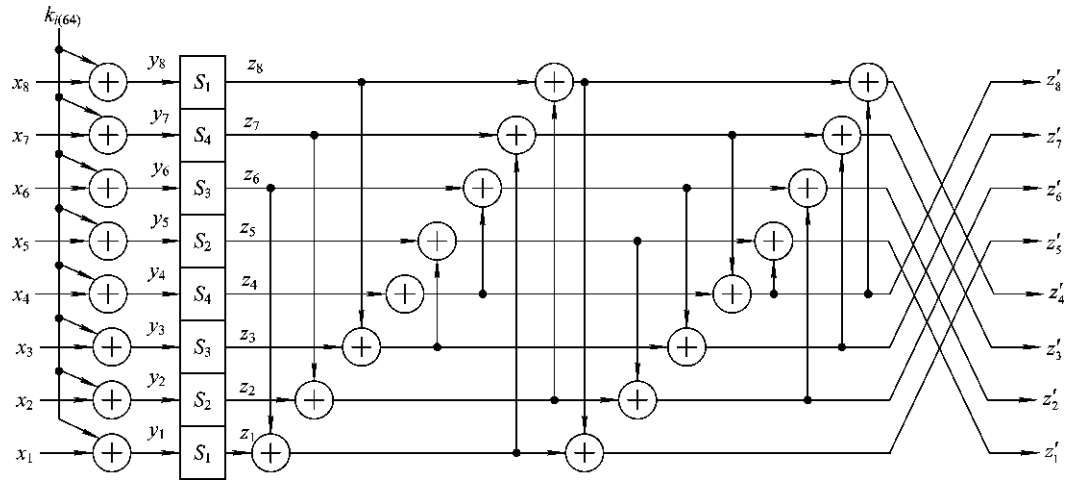

图 3.14 Camellia 算法的 F 函数

图中，64 bit 的数据 X 分成 8 组，各 8 bit；64 bit 的密钥也分成 8 组，各 8 bit，分别模 2 加，得到数据 Y；Y 进入 S 盒进行代替运算，得到数据 Z；再经线性变换 P 后得到 Z'。

2）S 盒

Camellia 组件总共用了 4 个不同的 8×8 双射 S 盒，排列次序见图 3.14。

$$S_1: y_{(8)} \rightarrow h(g(f(\mathrm{C5} \oplus y_{(8)}))) \oplus 6E \rightarrow z_{(8)} \tag{3-12}$$

$$S_2: y_{(8)} \rightarrow S_1(y_{(8)}) <<< 1 \rightarrow z_{(8)} \tag{3-13}$$

$$S_3: y_{(8)} \rightarrow S_1(y_{(8)}) >>> 1 \rightarrow z_{(8)} \tag{3-14}$$

$$S_4: y_{(8)} \rightarrow S_1(y_{(8)}) <<< 1 \rightarrow z_{(8)} \tag{3-15}$$

其中，f，g，h 均实现字节变换，即 $a_1a_2\cdots a_8 \rightarrow b_1b_2\cdots b_8$，具体为

$$\begin{aligned} f: & b_1 = a_6 \oplus a_2,\ b_2 = a_7 \oplus a_1,\ b_3 = a_8 \oplus a_5 \oplus a_3,\ b_4 = a_8 \oplus a_3 \\ & b_5 = a_7 \oplus a_4,\ b_6 = a_5 \oplus a_2,\ b_7 = a_8 \oplus a_1,\ b_8 = a_6 \oplus a_4 \end{aligned} \tag{3-16}$$

$$\begin{aligned} g: & (b_8 + b_7\alpha + b_6\alpha^2 + b_5\alpha^5) + (b_4 + b_3\alpha + b_2\alpha^2 + b_1\alpha^5)\beta \\ & = 1/((a_8 + a_7\alpha + a_6\alpha^2 + a_5\alpha^3) + (a_4 + a_3\alpha + a_2\alpha^2 + a_1\alpha^3)\beta) \end{aligned} \tag{3-17}$$

式中，β 是 $\mathrm{GF}(2^8)$ 上满足 $\beta^8+\beta^6+\beta^5+\beta^3+1=0$ 的元素；$\alpha=\beta^{238}=\beta^6+\beta^5+\beta^3+\beta^2$，是 $\mathrm{GF}(2^8)$ 中满足 $\alpha^4+\alpha+1=0$ 的元素。

$$\begin{aligned} h: & b_1 = a_5 \oplus a_6 \oplus a_2,\ b_2 = a_6 \oplus a_2,\ b_3 = a_7 \oplus a_4,\ b_4 = a_8 \oplus a_2 \\ & b_5 = a_7 \oplus a_3,\ b_6 = a_8 \oplus a_1,\ b_7 = a_5 \oplus a_1,\ b_8 = a_6 \oplus a_3 \end{aligned} \tag{3-18}$$

S 盒可以用硬件电路实现，也可以进行查表运算。

3）P 字符变换

P 字符变换为

$$Z' = PZ$$

其中：

$$P = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{3-19}$$

4）$\mathrm{FL/FL}^{-1}$ 函数

引入 FL 和 FL^{-1} 是为了提供轮间的不规则性，加强抗攻击性能。FL/FL^{-1} 仅由逻辑运算组成，如与、或、异或以及循环左移等，它的软、硬件实现都很快，如图 3.15 所示。

图 3.15 中 $\cap$ 为与运算；$\cup$ 为或运算；$\oplus$ 为异或运算；$\boxed{<<<1}$ 为循环左移 1 位的运算。

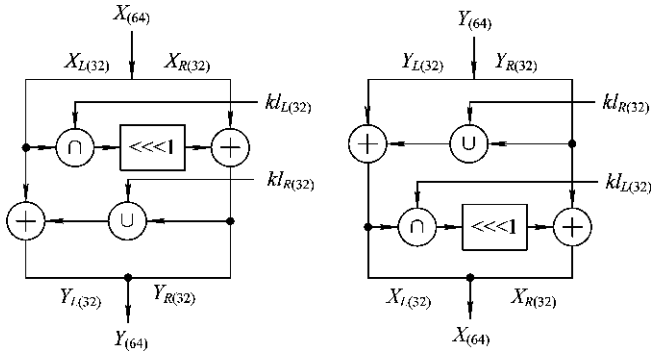

图 3.15 Camellia **算法的** FL **和** FL^{-1} **函数**

3.3.3 RC6 算法

RC6 **是** RSA **公司提给** NIST **的候选算法。它的全称是** RC6－$w/r/b$。$w=32$，**是明文分组的比特数，** $r=20$，**是加密轮数，** $b=16(24, 32)$，**是字节表示的密钥长。**RC6 **用了下列几种基本运算：**

(1) **整数模** 2^w **加和减，分别表示为**＋**和**－。

(2) **比特逐位模** 2 **加，表示为**⊕。

(3) **整数模** 2^w **乘，表示为**×。

(4) **循环左移** ROL、**循环右移** ROR **分别表示为**<<< **和** >>>。

1. 加密过程

把 128 bit **明文放入** 4 **个** 32 bit **的寄存器** A, B, C, D **中，** $S[i]$ $(i=0, 1, \cdots, 2r+3)$ **是** $2r+4$ **个子密钥，共经过** r **轮循环处理，其中第** i **轮的处理为**

FOR $i=1$ to r do

$t=\mathrm{ROL}(B\times(2B+1), \mathrm{lb}w)$

$u=\mathrm{ROL}(D\times(2D+1), \mathrm{lb}w)$

$A=\mathrm{ROL}(A \oplus t, u)+S[2i]$

$C=\mathrm{ROL}(C \oplus u, t)+S[2i+1]$

$(A, B, C, D)=(B, C, D, A)$

最后一轮之后，做 $A=A+S[2r+2]$, $C=C+S[2r+3]$**处理，**(A, B, C, D)**即为密文。**

RC6 **的加密过程如图** 3.16 **所示。**

2. 解密过程

128 bit **密文放入** 4 **个** 32 bit **的** A, B, C, D **中，** $A=A-S[2r+2]$, $C=C-S[2r+3]$, **仍须** r **轮循环处理：**

FOR $i=r$ to 1 do

$(A, B, C, D)=(D, A, B, C)$

$u=\mathrm{ROL}(D\times(2D+1), \mathrm{lb}w)$

$t=\mathrm{ROL}(B\times(2B+1), \mathrm{lb}w)$

$C=\mathrm{ROR}(C-S[2i+1], t) \oplus u$

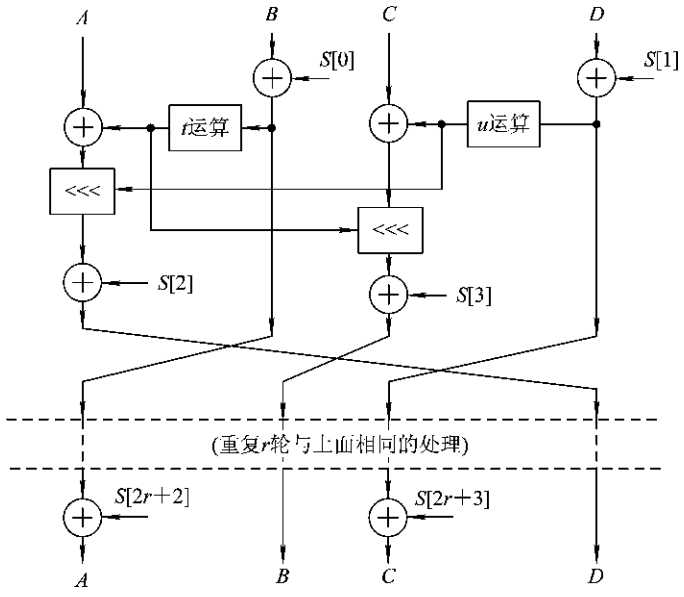

图 3.16　RC6 **的加密过程**

$$A=\mathrm{ROR}(A-S[2i],\ u)\oplus t$$

最后一轮之后，做 $B=B-S[0]$，$D=D-S[1]$**处理，**$(A,\ B,\ C,\ D)$ **即为明文。**

3. 子密钥的产生

令 $P_w=$B7E15163，$Q_w=$9E3779B9，$c=8b/w$ **的整数部分；首先将种子密钥** k **放入** c **个** w **比特的阵列** $L[0]$，$L[1]$，…，$L[c-1]$**中，不够补** 0**；再做如下处理：**

$$S[0]=P_w$$

FOR $i=1$ to $2r+3$ do

$$S[i]=S[i-1]+Q_w$$

$$A=B=i=j=0$$

$$v=3\times\max\{c,\ 2r+4\}$$

FOR $s=1$ to v do

$$A=S[i]=\mathrm{ROL}(A+B+S[i],\ 3)$$

$$B=L[j]=\mathrm{ROL}(A+B+L[j],\ A+B)$$

$$i=i+1 \bmod (2r+4)$$

$$j=j+1 \bmod c$$

输出的 $S[0]$，$S[1]$，…，$S[2r+3]$**即为子密钥。**

4. RC6 的特点

RC6 **有如下的特点：**

(1) 对内存要求很低(无需查表)，可以在单片机上实现，比如可在 IC **卡上用。**

(2) 抗攻击性好，用线性分析法至少需要 2^{155} **个明文，用差分分析法至少需要** 2^{238} **个明文。**

习 题 3

1. 在 DES 数据加密标准中，设主密钥 K(64 位)为

K=0000 0001 * 0010 0011 * 0100 0101 * 0110 0111 * 1000 1001 * 1010 1011 * 1100 1101 * 1110 1111 *

其中，* 号前面的码为奇偶校验位。请分别求出：

(1) 经 $PC-1$ 选定后的 C_0、D_0(56 bit)的结果。

(2) 经循环左移 $LS_1=1$ 位后的 C_1，D_1 的结果。

(3) 由 C_1、D_1 再经 $PC-2$ 选择后的 K_1(48 bit)的结果。

2. 已知输入为 011001 和 110010，根据 DES 的 S 盒的特殊性质，分别从 S_3 和 S_7 中求出压缩输出。

3. 在 IDEA 密码算法中，已知明文 m 和密钥 k 分别如下：

m=10101010 11110000 01010101 00001111 11001100 00110011 10101010 01010101

k=00000000 00000000 11111111 11111111 11111111 11111111 00000000 00000000 11110000 11110000 11110000 11110000 00001111 00001111 00001111 00001111

试求出第一轮变换后的输出和第二轮的输入。

4. 计算机存储口令的普遍方法是用户通过口令用 DES 加密一个固定的明文(比如 000000)，将密文存储在文件中。当用户再次登录时，询问口令并重复此过程，然后比较密文。为什么这种方法比直接核对口令或者存储加密的口令更安全？

5. 设分组长度为 128 bit，如果输入为“Alice just do it”，初始密钥使用 10101010 02020202 30303030 06060606，那么第一轮的 AES 输出是什么？

实践练习 3

实验题目： DES(或 AES)进行分组加密与解密练习。

实验平台： Turbo C。

实验内容： 对任意明文用 DES 或 AES 进行分组加密与解密。

实验步骤：

(1) 对任意长度明文进行分组。

(2) 把密钥与明文分组都变成二进制码。

(3) 实现一组的加密运算。

(4) 借助循环实现各组的加密。

(5) 实现密文的解密以验证加密过程。

第4章 公钥密码

计算机互联网的出现，极大地方便了人们对信息的交换和共享，电子商务和电子公务等越来越多的业务开始在网上进行，这就对信息交互提出了一系列新的需求。另一方面，网络也给攻击者提供了更多的机会，病毒、黑客以及网络犯罪时有发生。如何更好地解决信息安全问题，已成为刻不容缓的任务。20 世纪 70 年代出现的公开密钥体系，标志着现代密码学的诞生。新的理论与实践迅速发展起来了，随着一系列新观念和方法的出现，现代信息网络中的安全问题逐步得到解决，密码学进入了崭新的发展阶段。

4.1 引　　言[3]

4.1.1 对称密钥的困惑

1. 密钥交换问题

为了使对称密钥(单钥制)体系实现信息的保密传输，通信双方必须事先约定相同的算法和步骤，称之为保密协议，并且设法在不让其他任何人知晓的条件下，双方获取统一的密钥，这一过程叫做密钥交换。密钥交换往往是比较困难的任务，密钥是用秘密信道传递的，而秘密信道却昂贵且难得。因此，解决密钥交换问题便成了对称密钥体制的最大障碍。

2. 通信网络中的密钥管理问题

网络中 N 个用户两两保密通信，需要分配和保管 $M=\frac{1}{2}N(N+1)$ 把密钥，当 N 很大时，比如 $N=1000$ 时，则 $M\approx 5\times 10^5$，网管中心一定非常繁忙，甚至会成为通信的瓶颈。因此，密钥管理问题成了信息系统的又一难题。

4.1.2 公开密钥的产生和雏形

1. Merkle 难题

1974 年，Merkle 为解决对称单钥制保密系统的密钥交换问题提出了一个设想[9]。其密钥交换协议为：

(1) 甲向乙发送 100 万条消息，每条内容均为："这条信息是 x_i，它的密钥是 y_i"；x_i 是从 0 到 999 999 之间任意选取但又各不相同的随机数，y_i 是 x_i 的密钥，它也是从 0 到 999 999 之间的任意被甲预先指定的一个各不相同的随机数。甲秘密保存所有 y_i 与 x_i 的密钥对照表后，就把这 100 万条消息分别用所宣布的这个 y_i 加密，全部发给乙。

(2) 由于加密算法是公开的，乙收到100万条消息后，从中任选一条，然后遍历0～999 999之间的100万个密钥进行尝试，总有一个 y_j 可将其解密，从而得知 x_j 的值。

(3) 乙以明文形式将 x_j 发给甲。

(4) 甲收到 x_j，即知乙所选的是哪个 y_j，从而甲、乙二人都有了同样的密钥 y_j。

(5) 非法窃听者丙即使收到了全部往来的明文和密文，虽然知道了 x_j，但他不知道 x_j 在哪条消息中，他需要对100万条消息一一进行解密，而每解译一条消息需要尝试100万个密钥。假若乙耗时半分钟能完成对100万个密钥的尝试，得到自己的密钥，丙则需要100万倍的时间，大约1年才能从100万条消息中确定乙所选的是哪一条。

此设计方案显然是不太现实的，但其思想是很先进的。它用公开的算法通过不安全信道完成了密钥交换，其保密理念基于计算复杂性，安全性则是建立在机密与破译的时差之上的。该课题的出发点虽然是为了解决单钥交换问题，但客观上已走进了公开密钥的大门。

2. 双钥制的提出

1976年11月，美国斯坦福大学电气工程系研究生 W. Diffie 和副教授 M. E. Hellman 在 IEEE 上发表了题为“密码学新方向”的学术论文[10]，首次提出双密钥思想，用公开的算法解决了单钥制的密钥交换问题。

设 p 为一个大素数，已知 x 和 b，不难求出：

$$y = b^x \bmod p \tag{4-1}$$

然而若已知 y 和 b，求逆运算：

$$x = \log_b y \bmod p \tag{4-2}$$

却十分困难。利用离散对数复杂性，他们二人设计的密钥交换协议如下：

(1) 用户 i 选 x_i 为私钥，求出 $y_i = b^{x_i} \bmod p$ 为公钥，公布之(连同 b 和 p 发给用户 j)。

(2) 用户 j 选 x_j 为私钥，求出 $y_j = b^{x_j} \bmod p$ 为公钥，公布之(发给用户 i)。

(3) 约定

$$k_{ij} = b^{x_i x_j} \bmod p \tag{4-3}$$

为会话密钥(对称单钥)，用户 i 可由

$$k_{ij} = (b^{x_j})^{x_i} \bmod p = (y_j)^{x_i} \bmod p$$

获得；用户 j 可由

$$k_{ij} = (b^{x_i})^{x_j} \bmod p = (y_i)^{x_j} \bmod p$$

获得。

(4) 尽管公布了 y_i、y_j 和 p，但基于离散对数的困难性，任何第三者都难以求出 x_i 和 x_j，因此无法获得 k_{ij}。

Diffie 和 Hellman 不仅用公开的算法，而且首次提出了公钥和私钥的概念，开创了现代密码学的先河。

4.1.3 公开密钥制的一般原理

1. 公开密钥体系的使用方式及功能

W. Diffie 和 M. E. Hellman 的论文发表后，立即引起了学术界的重视，并且很快把双

密钥的思想用到密码系统中，这不仅解决了单钥制的密钥交换问题，更重要的是以一种全新的方式很好地解决了保密、认证与网络管理等问题。

假设在算法公开的双密钥密码体制下每个用户都分得了自己的公钥(可以公开的密钥)与私钥(必须秘密保存的密钥)，于是就能完成以下功能：

(1) 保密功能(包括会话密钥的交换)：发信人用收信人的公钥加密，形成密文，收信人用自己的私钥解密。其他人因为不掌握可配对的私钥，所以无法解读这个密文。

(2) 认证功能：发信人用自己的私钥加密形成密文，收信人用发信人的公钥解密。其实任何人都能查到发信人的公钥来解译密文，能正确解译即证明该密文是发信人所加密的。

(3) 双重功能：发信人先用自己的私钥加密明文，再用收信人的公钥对密文再次加密；收信人收到密件后先用自己的私钥解密一次，再用发信人的公钥解密一次。这样的密文只有合法收信人才有能力阅读并验证。

(4) 网管功能：N 个用户的系统，管理员只需保管(不必隐藏)N 把公钥，私钥由用户个人保管。密钥分配与管理的问题立刻变得简单了。

2. 公开密钥体系的加、解密过程

公开密钥体系把算法公开，并且把一对密钥中的一把公开，才换来了功能上的增加与使用上的方便。而之所以将之公开，是因为解密所用算法并非加密的逆运算，解密所用密钥也不同于加密所用的密钥，而且二者不可互相推导。

加密：明文 M→变换 E_{k_1}(k_1 为加密密钥)→密文 C，即

$$C = E_{k_1}[M] \tag{4-4}$$

解密：密文 C→变换 D_{k_2}(k_2 为解密密钥)→明文 M，即

$$M = D_{k_2}[C] \tag{4-5}$$

3. 单向陷门函数(加密、解密的数学原理)

作为一个保密系统，无论单钥制还是双钥制，解密和加密的效果一定应当是互逆的，通过解密应得到与原来的明文相同的译文。一般说来，解密应当用加密的逆运算实现，单钥制就是这么做的。然而，抛开逆运算的定式看问题，逻辑上并不排除存在其他算法也能解出正确译文的途径。如果有，为什么不可以用呢。至于密钥，它只是加、解密运算过程中的一些关键数据，更没有理由认为参与加密运算的密钥和参与解密运算的密钥必须相同。(当然，加密密钥与解密密钥既然是配对的，那么总应当存在某种内在的联系，实际上也确实如此。不过只要这种内在联系涉及到复杂度很高的运算，就可以认为它们是无法相互推导的)。于是，问题又回到了数学上，能否找到这样的算法和密钥呢？

数学上存在一种单向陷门函数，它有下列性质：

(1) 对于每个 $x\in X$，计算 $y=f(x)$是容易的。正是因为这一点，加密运算才是简单可行的。

(2) 明知 $y=f(x)$在 $y\in Y$ 中有解，但由给定的 y 难以求出 x，其逆运算 $x=f^{-1}(y)$ 十分复杂。正是因为这一点，它被叫做单向函数。即使公开了算法，也难以在有限机时内计算出逆运算结果，系统安全性才有了保障。

(3) 对于某些特殊的单向函数(不是所有的单向函数)，若附加一点相应的“陷门信息

k”，则存在另一个能算出 x 的方法：$x=f_k(y)$。$f_k(y)$不同于 $f^{-1}(y)$，它可以容易地算出 x；求解这个问题的关键数据 k 就是密钥。正是因为这一点，掌握密钥的合法用户才能容易地正确解译密文。

数学中确实存在不少逆运算非常复杂的单向函数，如大数的分解因数、离散对数等等，这是构造公开密钥体系的必要条件。但是还必须设法引入“陷门”，就像魔术师做一个套，知道窍门的人，就能轻易地从别的路钻出来，而不必按进来的路线在迷宫里摸索。目前已找到多种单向陷门函数，设计出多种公开密钥系统，如 RSA 系统、背包系统、ElGamal 系统、Rabin 系统等，后面将陆续介绍。

4.1.4 现代密码学的理念

1. 从基于算法的神秘性到基于算法的复杂性

传统密码体系的安全性依赖于对算法的保密，一旦算法失密，攻击者就可以用它的逆运算来破译密码系统。而现代密钥学则利用逆运算复杂度非常高的单向算法来构建密码系统，就不必担心算法失密，因为攻击者即使应用现代最新计算技术，在有限时间内也是无法算出结果的。

所谓复杂性，是指计算所必须的基本运算步骤的数量，它决定了计算所必须的机时和占用的计算机资源。计算复杂性理论告诉我们，计算量随着数据位数 N 的增长而变大，其增大的速度与算法的函数类型有关。按照从小到大的次序是：常数，对数函数 $\log N$，线性函数 aN，二次函数 N^2，三次函数 N^3，多项式函数 N^d，亚指数函数 $2^{\lg N}$，指数函数 2^N 或 10^N。

随着数据位数 N 的增长，计算量按照多项式以下的依赖关系变大的算法都可认为是有效算法，这样的增加速度对于现有计算技术来说是可以接受的，有限时间内能够算出结果。否则，计算量将随着 N 的增长而迅速膨胀，就不可能在有限时间内算出结果。复杂性理论把依赖关系为多项式及其低于多项式的算法都称为可解问题（p 类），而将超过多项式的算法都称为难解问题（Np 类，NpC 类）。

如果已经证明了某种密码的破译是 Np 类问题，那么只要数据位足够大，则任何现代的计算设备都不可能在允许的时间内将其破译，因此它是安全的。由基于算法的神秘性到基于算法的复杂性是现代密码学设计理念上的一次重大转变，不靠神奇靠麻烦，算法不怕别人猜出，甚至可以公开。这样一来，密码分析者的注意力也就从研究由密文提取信息的可能性（老观点）改变为由密文提取信息的可行性（新观点）。

2. 算法的可公开性

现代密码学认为藏着捂着的神秘算法，其安全性终究是侥幸的和脆弱的，只有根据算法的复杂性建立起来的密码体制，其安全性才是坚固可信的。这是因为算法的复杂性完全能够通过数学理论科学地预测，只要该算法复杂到不可行的程度，即使公开了算法，也难以在有限机时内计算出逆运算结果。算法既然失去了保密的价值，公开算法就成了现代密码体制的一大特点。算法公开后，可以让它在攻击中不断完善和改进，并以此显示其安全的坚固性。

3. 安全的相对性

在科学技术高度发展的今天，应充分估计破译者的计算能力和计算技术未来的发展，

从这个意义讲，不存在永远牢不可破的密码，只存在当前阶段与需求相适应的安全密码体系，破译只是时间和金钱的问题。但是如果破译工作所花的代价大于秘密本身的价值，或破译花费的时间大于秘密的有效期，则破译失去意义，则该保密系统就可以认为是安全的。这才是实事求是的科学的保密观和破译观。根据这个观点，破译的工作量取决于算法的复杂度，而复杂性又取决于数据位数的长短，因此可以根据客观需求实事求是地设计不同层次、不同时效的密码体系。不求绝对(安全)求相对(安全)是密码设计理念上的又一个转变。

4. 密钥的机密性

算法公开了，合法用户与非法用户的区别在哪里呢？合法用户拥有密钥，解译密文十分容易；非法用户没有密钥，破译密文则很不容易。这是因为如果攻击者用遍历法一个个去尝试密钥，设每尝试一个密钥需要 1 秒钟，则遍历 160 比特长的所有密钥需要 2^{160} 秒钟，约 10^{40} 年。只要密钥具有足够的长度与随机性，偶然猜中密钥的概率是极小的。不藏算法藏密钥是设计理念上一致的观点。保守密钥的机密要比隐藏算法的机密容易得多，也安全得多。况且密钥是可以随时更换的，万一暴露了密钥，可以换一把，不致造成整个系统的破坏。

4.2 背包公钥密码系统

当初 Diffie 和 Hellman 提出公钥思想的时候，还没有一个实例。两年后，1978 年，Merkle 和 Hellman 利用古老的背包问题设计出一个公开的密钥系统[11]。

4.2.1 背包问题

1. 问题

已知 n 个物体重量分别为 $\boldsymbol{A}=(a_1, a_2, \cdots, a_n)$，任选其中若干件放入背包内，使其总重量恰好等于预先给定的 b 值。

设所选物体用 $\boldsymbol{X}=(x_1, x_2, \cdots, x_n)$ 表示，这里 $x_i(i=1, 2, \cdots, n)$ 可为 0 或 1，分别表示该物体被取或不被取，则

$$\boldsymbol{A} \cdot \boldsymbol{X} = \sum_{i=1}^{n} a_i x_i = b \tag{4-6}$$

2. 解答

当 n 较小时，这是一个多解的不定方程问题。例如，$A=\{1, 2, 5, 8, 11, 17, 21\}$，$b=47$，则答案可以是 $2+5+8+11+21$，也可以是 $1+8+17+21$。

当 n 较大时，它是一个 Np 类的难解问题。例如，$n=100$，$2^{100}=1.27\times10^{30}$，以每秒 10^7 种方案的速度用计算机搜索，也得 4.02×10^{15} 年才能完成穷举。

3. 超递增序列的背包问题

如果限制这些物体的重量，即每个物体的重量都满足比它的前面所有物体的重量之和大的条件，即

$$a_i > \sum_{j=1}^{i-1} a_j \tag{4-7}$$

则这样的背包问题叫做超递增背包问题。超递增背包问题是 p 类唯一解的可求解问题。

例如：$A=\{1, 2, 5, 10, 25, 51\}$，$b=64$，则由 $64>51$ 可知 a_6 必存在；再由 $64-51=13<25$ 知 a_5 不必取，而由 $13>10$ 知 a_4 必选；又由 $13-10=3<5$ 知 a_3 必没有；而又由 $3>2$ 知 a_2 必选；最后由 $3-2=1=a_1$ 知 a_1 也必选。故答案是 $\boldsymbol{X}=\{1, 0, 1, 0, 1, 1\}$，即 $64=1+2+10+51$。

4.2.2 MH背包公钥系统

设明文为 $\boldsymbol{M}=\{m_1, m_2, \cdots, m_n\}$，$m_i=\begin{cases}0\\1\end{cases}$。若用超递增序列 $\boldsymbol{B}=\{b_1, b_2, \cdots, b_n\}$ 将之加密，则 $C'=\sum_{i=1}^{n} m_i b_i$ 是 p 类可解问题；若用非超序列 $\boldsymbol{A}=\{a_1, a_2, \cdots, a_n\}$ 将之加密，则 $C=\sum_{i=1}^{n} m_i a_i$ 是 Np 类难解问题。

如果设计出一个方法，能把 $\boldsymbol{B}$ 变换成 $\boldsymbol{A}$，我们用 $\boldsymbol{A}$ 来加密，使问题成为 Np 类完全难题，则合法用户由于掌握 $\boldsymbol{A}$ 变换成 $\boldsymbol{B}$ 所具有的信息(私钥)，就能由 $\boldsymbol{A}$ 求出 $\boldsymbol{B}$，使问题成为 p 类可求解问题。而非法用户只知道 A(公钥)，不掌握私钥，因此就无法解答此题。

Merkle 和 Hellman 当初是利用模逆元进行这个变换的。例如，$\boldsymbol{B}=\{1, 3, 7, 13, 26, 119, 267\}$，选大于全部元素之和 501 的一个数，$p=523$，再选一个与 523 互素的数 $w=28$，并求出 $w^{-1}=467 \bmod 523$，于是可分别求出每个 b_i 的模逆元：

$$a_i = w^{-1} \cdot b_i \bmod 523 \quad (i = 1, 2, \cdots, n)$$

结果是：

$$\begin{aligned}\boldsymbol{A} &= \{467\times 1, 467\times 3, 467\times 7, 467\times 13, \cdots\} \bmod 523\\ &= \{467, 355, 131, 318, 113, 21, 135, 215\}\end{aligned}$$

$\boldsymbol{A}$ 和 $p=523$ 为公钥，求出 A 后，将 w^{-1} 销毁。要发送信息就用 $\boldsymbol{A}$ 来加密。

例如，明文 $M=10101100$，则密文为

$$C = \boldsymbol{A}\cdot\boldsymbol{M} = a_1 + a_3 + a_5 + a_6 = 467 + 131 + 113 + 21 = 732 \bmod 523 = 209$$

$w=28$ 为合法用户的私钥。合法接收者不难求出：

$$b_i = wa_i \bmod p = w \cdot w^{-1} b_i \bmod p = b_i \bmod p (i = 1, 2, \cdots, n)$$

即求出

$$\boldsymbol{B} = \{1, 3, 7, 13, 26, 65, 119, 267\}$$

还能求出

$$C' = wc \bmod p = 28 \times 209 \bmod 523 = 99$$

而这是一个超递增背包问题，容易解出：

$$m_1 = m_3 = m_5 = m_6 = 1$$

其他 $m_i=0$，所以明文是 $M=10101100$。

4.2.3 背包公钥体系的破解与发展现状

Merkle 和 Hellman 提出背包体系时，曾悬赏 50 美元征求人破译，两年后，Shamir 将

其破译了。尽管求大数的模逆元与分解大素数的复杂度相同，但该设计存在漏洞。后来 Merkle 和 Hellman 将漏洞补上，再次悬赏破译，两年后又被人破译，使背包体系受到致命打击。

背包体系目前基本上已不大用了，但它作为公钥的先驱实践者，作为算法和思路很简单的公钥体制，仍有了解的必要。同时，以背包问题为原理的各种新密码体制目前仍有人在继续研究。

4.3 RSA 公钥密码(基于大数分解)[3, 7, 8]

RSA 公钥密码是第一个实用的，同时也是流行至今的最典型的公钥算法。1978 年，美国麻省理工大学的 Rivest、Shamir 和 Adelman 利用数论相关知识，提出了这个公钥系统[10]。

4.3.1 RSA 加解密原理

设 p 和 q 为两个大素数，计算 $n=pq$ 和欧拉数 $\Phi(n)=(p-1)(q-1)$，随机选择整数 e，使满足$(e, \Phi(n))=1$，于是它的逆元存在，即 $d=e^{-1} \bmod \Phi(n)$，即有：

$$ed = 1 \bmod \Phi(n) \quad 或 \quad ed = k\Phi(n)+1 \tag{4-8}$$

设 m 为明文，e 为公钥，n 也是公钥(公钥是两个数据)，则加密过程为

$$C = m^e \bmod n \tag{4-9}$$

因为逆运算 $m=\sqrt[e]{\lambda n+C}$ 十分复杂，且存在多义性(λ 不定)，所以它是一个单向函数。

然而，利用欧拉定理知，若 m 与 n 互素，则

$$m^{\Phi(n)} = 1 \bmod n \tag{4-10}$$

对任意整数 k，m^k 仍与 n 互素，则

$$(m^k)^{\Phi(n)} = 1 \bmod n$$

于是：

$$m^{k\Phi(n)+1} = m \bmod n$$

由此得到解密算法：

$$C^d = m^{ed} \bmod n = m^{k\Phi(n)+1} \bmod n = m \bmod n$$

即

$$m = C^d \bmod n \tag{4-11}$$

这里，d 为私钥，只有合法用户掌握；p、q 和 $\Phi(n)$ 在完成设计后都可销毁；仅由公钥 e 和 n 是无法求出 d 的，除非能将 n 分解，求出 p 和 q，而这是 Np 类难题，难以实现。

4.3.2 RSA 的参数选择

1. p 和 q 应为强素数

强素数是这样一种素数：对于素数 p，若存在素数 p_1 和 p_2，使 $p_1 | p-1$，$p_2 | p+1$，则称 p 为一级素数，称 p_1 和 p_2 为二级素数；对于 p_1 和 p_2，若存在素数 $r_1 r_2$ 和 $s_1 s_2$，使

$r_1 \mid p_1-1$，$s_1 \mid p_1+1$，$r_2 \mid p_2-1$，$s_2 \mid p_2+1$，则称 r_1、r_2、s_1、s_2 为三级素数。存在二级和三级素数的一级素数才是强素数。这样才能保证在有效时间内不可能将其分解。反之，就有可能找到一些简便方法将其分解。

2. p 和 q 的位数问题

提出 RSA 的三人最初建议 p 和 q 为 100 位的十进制数($\approx 2^{332}$)，这样可使 n 达到 200 位以上，在亿次机上分解需 55 万年。同时，他们希望 p 和 q 的位数相差一般是几比特，这是因为如果相差太小，$\tau=\dfrac{p+q}{2}$ 就接近于$\sqrt{n}$，而 $h=\dfrac{p-q}{2}$就很小，从而使费尔玛因式分解法容易进行：

$$\tau^2-h^2=\frac{1}{4}[(p+q)^2-(p-q)^2]=pq=n \tag{4-12}$$

从小数的平方来测试，可以大大减少复杂性。

3. e 和 d 的选择

e 不能太小，太小了可以由$\sqrt[e]{C}$得到 m。d 小了虽然有利于解密的速度，然而 d 太小了也存在隐患，破解者可以对 C^d 穷举以获得 m。因此，最好 $d \geqslant n^{\frac{1}{4}}$。

4. n 在使用上的限制

不宜用同一个 n 去设计若干对(e, d)，这样会引起不安全因素。假若对同一明文 m：

$$C_1=m^{e_1} \bmod n, \quad C_2=m^{e_2} \bmod n$$

这里 e_1，e_2 互素，即满足 $te_1+se_2=1$，于是有：

$$C_1^t C_2^s=m^{te_1}m^{se_2}=m^{te_1+se_2}=m \bmod n$$

这样一来，无需私钥，就得到了明文。

4.3.3 素数的检测

1. 素数是否够用

数论有定理指出，对正整数 N，小于 N 的素数数目约为$\dfrac{N}{\ln N}$。若 p，q 为十进制 154 位(512 bit)的大素数，那么在此范围内约有 10^{151}个素数。宇宙中原子仅 10^{77}个，如果每个原子从宇宙诞生至今每秒钟需要一个素数，也才 10^{109}个。

2. 是否会两人偶然巧合选择了同一个素数

小于 n 的素数的个数是$\dfrac{n}{\ln n}$，从中任选一个素数的概率是$\dfrac{\ln n}{n}$；当 n 是 N 位十进制大数时，这个概率小于$\dfrac{N\ \ln 10}{n}$，可见这种巧合的概率几乎为零。

3. 能否建立所有素数的数据库，用来破译 RSA(分解 n)

理论上可以这样做，但实际行不通。设每克半导体存储器可存储 10 亿字节，将 512 位的全部素数集中在有限尺寸的内存中，其存储器质量将超过引力场极限，使它变成黑洞，无法提取数据。

4. 怎么知道一个数是否为素数

不能再分解的数才是素数，为此就得将其分解。而分解因数是 Np 问题，正因为它无

法办到所以才有了 RSA 的安全性。那 RSA 所用的素数从何而得呢？

似乎像是个先有鸡还是先有蛋的驳论。然而，检测素数毕竟与分解因数不同。现已找到多种素数测试方法和理论。

① 威尔逊定理：n 为素数的充要条件是$(n-1)!\equiv -1 \bmod n$(见前)。

② 费尔马定理：若 p 为素数，则对任一整数 a，有 $a^{P-1}\equiv 1 \bmod p$，但这只是必要条件。反过来，满足费尔马定理的并不一定都是素数，如 $n=561=3\times 11\times 17$ 就满足 $a^{n-1}\equiv 1 \bmod n$，这样的 n 被称为卡密沙尔数(Carmichael)。

③ 欧拉定理：二次剩余理论中指出，p 是素数的充要条件是 $a^{\frac{p-1}{2}}\equiv 1 \bmod p$，这里 a 是 p 的二次剩余。若 a 非二次剩余，则 $a^{\frac{p-1}{2}}\equiv -1 \bmod p$；若 p 是 a 的倍数，则 $a^{\frac{p-1}{2}}\equiv 0 \bmod p$。合起来就是勒让德函数：

$$L(a,\ p)=a^{\frac{p-1}{2}} \bmod p=\left(\frac{a}{p}\right)=\begin{cases}1\\0\\-1\end{cases} \tag{4-13}$$

当不清楚 n 是不是素数时，上式为雅可比函数：

$$J(a,\ n)=a^{\frac{p-1}{2}} \bmod n \tag{4-14}$$

设 $n=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$，则

$$J(a,\ n)\equiv J\left(\frac{a}{n}\right)=\left(\frac{a}{p_1}\right)^{\alpha_1}\left(\frac{a}{p_2}\right)^{\alpha_2}\cdots\left(\frac{a}{p_k}\right)^{\alpha_k}=\begin{cases}1\\0\\-1\end{cases} \tag{4-15}$$

由此得到素数判别程序的算法描述如下：

S1：随机产生一整数 $a\in[1,\ n-1]$。

S2：计算 $\gcd\{a,\ n\}$。

S3：若 $\gcd\{a,\ n\}\neq 1$，则 n 非素数，结束。

S4：否则计算$\left(\frac{a}{n}\right)$及 $a^{\frac{n-1}{2}} \bmod n$。

S5：若$\left(\frac{a}{n}\right)=a^{\frac{n-1}{2}} \bmod n$，则 n 可能是素数，转 S1，进行第二轮，否则 n 是合数，结束。

这个算法每进行一轮测试，正确性至少为$\frac{1}{2}$；判断错误的概率小于$\frac{1}{2}$；若两轮检测都通过，则判错概率小于$\left(\frac{1}{2}\right)^2$；若 n 轮检测都通过，则判错概率小于$\left(\frac{1}{2}\right)^n=\frac{1}{2^n}$。进行 $n=10\sim 20$ 轮判断，可使判错概率小于万分之一。这种素数称为工业级素数。

④ 米勒-列宾(Miller-Rabin)测试法：使用方法③收敛很慢，而使用 M-R 法 k 次通过的判错率仅为$\left(\frac{1}{4}\right)^k=\frac{1}{2^k}\cdot\frac{1}{2^k}$，且比③减少一半的轮数。判别程序的算法描述如下：

S1：先化 $n-1=2^l m$（m 为去掉二进制最末位 1 后，再去掉 l 个连 0)。

S2：在$\{1,\ 2,\ \cdots,\ n-1\}$中随机均匀地产生数 b；$j\leftarrow 0$，计算 $z\leftarrow b^m \bmod n$。若 $z=1$ 或 $n-1$，则 n 通过测试，结束。

S3：若 $j=l$，则 n 非素数，结束，否则转 S4。

S4：$j=j+1$，$z\leftarrow z^2 \bmod n$。若 $z=-1$，则 n 通过测试，结束；否则转 S3。

当 n 通过第一轮测试后应回到S2，重新产生随机数 b，开始下轮测试。

4.3.4 RSA的安全性

RSA的安全性是基于大数进行因数分解的复杂性的。理论上已证明分解大数的渐进复杂度大约正比于 $e^{\ln n \ln(\ln n)}$，可见 n 必须足够大，分解才能足够复杂。

最初Rivest悬赏100美元破译129位(429比特)的RSA密码体系，估计百万次的计算机需连续工作4600年，结果43国600多人动用1600台计算机通过因特网联合工作，耗时8个月，于1994年4月20日破译。后来130位的RSA也于1996年4月10日，用数域筛选法被攻破。人们又向154位发起攻击。200位(664比特)和1024比特的RSA已有实用产品。

RSA的安全漏洞：由于双钥制既能用于加密，又能用于认证，当A用自己的私钥对某文档"加密"后，$y=m^d \bmod n$，y 就成为A对文档 x 的签名，任何人可以用A的公钥将其"解密"，即 $x=y^e \bmod n$，得到明文，同时证明该文是A所签发的。

利用这一点，窃密者想解析某人发给A的密文 $C=m^e \bmod n$，他可以先自己随机产生一个文档 r，用A的公钥加密为 $s=r^e \bmod n$，并求出 r 在模 n 下的逆 r^{-1}；之后，他把 $y=sC$ 发给A，请A签名，如果A同意签名，便计算 $u=y^d \bmod n$，发给窃密者；窃密者得到 u 后，计算 $r^{-1}u=r^{-1}y^d=r^{-1}s^dC^d \bmod n$，因为 $r=s^d \bmod n$，而 $r^{-1}s^d=r^{-1}r=1 \bmod n$，所以 $r^{-1}u=C^d \bmod n=m$，明文 m 被窃得。

这个攻击过程中，A被窃的疏漏在于他对陌生人的文档 y 进行了签名。为了骗取A的签名，有多种手法，比如A不愿对 m_3 签名(或许因 m_3 中有不利于A的文字)，骗子可以写 $m_3=m_1m_2$，而 m_1 和 m_2 的文字对A有利，它骗得 m_1^d 和 m_2^d 后，就可以计算 $m_3^d=m_1^dm_2^d$。又如，他还可以将 m_3 改头换面为 $y=m_3s$($s=r^e \bmod n$，是随机明文 r 的密文)，送 y 去让A签名，如果A签名了，则 $u=y^d \bmod n$，骗子便可以计算出：

$$y^dr^{-1}=m_3^ds^dr^{-1}=m_3^drr^{-1}=m_3^d \bmod n$$

因此，绝不要对一个陌生人交给你的消息进行签名。即使要签，也应当只对消息的HASH值签名。

4.4 Rabin公钥体系(基于二次剩余)

Rabin公钥体制是RSA的 $e=2$ 的特例。其加密算法是：

$$C=m^2 \bmod n \qquad (n \text{ 为公钥}) \tag{4-16}$$

它相当于同时满足：

$$C=m^2 \bmod p \quad \text{和} \quad C=m^2 \bmod q$$

表明 C 是两个素数共同的平方剩余：

$$\begin{cases} m^2=C \bmod p \\ m^2=C \bmod q \end{cases} \tag{4-17}$$

解密过程则是求同时满足模 p 和模 q 下密文 C 的平方根(p 和 q 是私钥)。

4.4.1 GF(p)上平方根问题的讨论

在GF(p)域就是在整数[1，$p-1$]范围内求解。根据数学补充中关于二次剩余的知识，不难知道在[1，$p-1$]中的平方剩余方程：

$$x^2 = a \bmod p$$

退化为

$$x^2 = a \qquad x \in [1, p-1]$$

费尔马定理 $a^{p-1} = 1 \bmod p$，则退化为

$$a^{p-1} = 1 \qquad x \in [1, p-1]$$

于是

$$a^{p-1} - 1 = 0 \Rightarrow (a^{\frac{p-1}{2}} - 1)(a^{\frac{p-1}{2}} + 1) = 0$$

表明在[1，$p-1$]上，有$\frac{p-1}{2}$个根满足 $a^{\frac{p-1}{2}} = 1 \bmod p$，它们是平方剩余(QR)；而另外$\frac{p-1}{2}$个根满足 $a^{\frac{p-1}{2}} = -1 \bmod p = p-1 \bmod p$，它们是非平方剩余(NQR)。

1. $\frac{p-1}{2}$是奇数的情况

设 β 是一个平方剩余，满足 $\beta^{\frac{p-1}{2}} = 1$，则

$$\beta^{\frac{p+1}{2}} = \beta^{\frac{p-1}{2}} \cdot \beta = 1 \cdot \beta = \beta \tag{4-18}$$

因为$\frac{p-1}{2}$为奇数，所以$\frac{p+1}{2}$必为偶数，所以$\frac{p+1}{4}$必存在，因此有

$$\beta = \beta^{\frac{p+1}{2}} = [\beta^{\frac{p+1}{4}}]^2$$

可见，$\alpha = \beta^{\frac{p+1}{4}}$ 是 β 的一个平方根。

另一个平方根是 $p-\alpha$，这是因为

$$(p-\alpha)^2 = p^2 - 2p\alpha + \alpha^2 = \alpha^2 \bmod p$$

结论：对于 $p \neq 2$ 的素数，它的平方剩余 β 有两个根：

$$\alpha = \beta^{\frac{p+1}{4}} \quad \text{和} \quad \alpha = p - \alpha \tag{4-19}$$

【例1】 $\beta = 2$ 是否为二次剩余方程为 $x^2 = \beta \bmod 23$ 的平方剩余？如果是，求方程的根。

解：由 $\beta^{\frac{p-1}{2}} = 2^{11} \bmod 23 = 2048 \bmod 23 = 1$ 知 $\beta = 2$ 确为平方剩余，且 $\frac{p-1}{2} = 11$ 为奇数。

于是模 23 运算下 $\beta = 2$ 的两个根是：

$$\alpha_1 = 2^{\frac{p+1}{4}} = 2^6 \bmod 23 = 18$$

和

$$\alpha_2 = 23 - 18 = 5$$

2. $\frac{p-1}{2}$为偶数的情况

令 $p-1 = 2^h q$，q 为奇数($p-1$ 被 2 连除 h 次才能得到奇数 q)，可以证明以下几条：

(1) 若 α 是 NQR，则 $r=\alpha^q$ 满足 $r^{2^h}=1 \bmod p$。

(2) $r^{2^{k-1}}=-1 \bmod p$。

(3) 若 $\beta\in$ QR，则存在偶数 $j(0\leqslant j\leqslant 2^h)$ 使 $\beta^q=r^j$。

(4) $r^{-j}\beta^{q+1}=\beta$。

(5) 令 $j=2k(0\leqslant k\leqslant 2^{k-1})$，则

$$(\beta^q)^{2^{k-2}}=\begin{cases}1 & \text{若 } k \text{ 为偶数}\\ -1 & \text{若 } k \text{ 为奇数}\end{cases}$$

证明：(1) 因为 α 假定为 NQR，$\alpha^{\frac{p-1}{2}}=-1$，所以

$$(\alpha^q)^{2^h}=\alpha^{p-1}\equiv 1 \bmod p$$

(2) 因为 $\alpha^{\frac{p-1}{2}}=-1 \bmod p$，而 $r=\alpha^q$，$p-1=2^hq$，$\frac{p-1}{2}=2^{h-1}q$，所以

$$\alpha^{\frac{p-1}{2}}=(\alpha^q)^{2^{h-1}}=r^{2^{h-1}}=-1 \bmod p$$

(3) 由于 $\beta\in$ QR，因此

$$\beta^{\frac{p-1}{2}}=(\beta^q)^{2^{h-1}}=1 \bmod p$$

又由于(2)的结论，因此若 j 为偶数，则

$$(r^{2^{h-1}})^j=(r^j)^{2^{h-1}}\equiv 1 \bmod p$$

比较可见：

$$\beta^q=r^j \qquad (0\leqslant j\leqslant 2^h)$$

(4) 由 $\beta^q=r^j$ 就有 $r^{-j}\beta^q=1$，即

$$r^{-j}\beta^{q+1}=\beta$$

或写为

$$(r^{-j/2}\beta^{(q+1)/2})^2=\beta$$

表明 $(r^{-j/2})\beta^{(q+1)/2}$ 是 β 的一个平方根。再利用(1)的结果就有

$$\beta^{\frac{q+1}{2}}\cdot r^{-\frac{j}{2}}=\beta^{\frac{q+1}{2}}\cdot r^{2^h-j/2}$$

(5) 令 $j=2k$，则 $\beta^q=r^j=r^{2k}$，所以

$$(\beta^q)^{2^{h-2}}=(r^{2k})^{2^{h-2}}=(r^{2^{h-1}})^k=(-1)^k=\begin{cases}1 & k \text{ 为偶数}\\ -1 & k \text{ 为奇数}\end{cases}$$

利用以上性质，得到计算 β 平方根的算法如下：

S1：通过测试 $\alpha^{\frac{p-1}{2}}=p-1 \bmod p$，来选择 $\alpha\in[1,\ p-1]$ 为 NQR。

S2：取值 $r\leftarrow\alpha^q$，$\delta\leftarrow\beta^q$，$I=h$，$J=1$，$K=0$。

S3：若 $\delta^{2^{I-1}}=J^{2^{I-1}}$，则转 S5。

S4：$J\leftarrow J\cdot(r)^{2^K}$。

S5：若 $I=2$ 则作：$r\leftarrow J^{-1}\beta^{(q+1)/2}$，$r$ 即 β 的平方根，结束。

S6：$I\leftarrow I-1$，$K\leftarrow K+1$，转 S3。

3. n 非素数，特别当 $n=pq$(p，q 均为素数)时 $x^2=b \bmod n$ 的讨论

$x^2=b \bmod n$ 等价于两个联立方程：$x^2=b \bmod p$ 和 $x^2=b \bmod q$。

(1) 当 $\frac{p-1}{2}$ 和 $\frac{q-1}{2}$ 均为奇数的情况。

由于它们分别有$\frac{p-1}{2}$和$\frac{q-1}{2}$个 QR，因此原方程有$\frac{1}{4}(p-1)(q-1)$个 QR。

两个方程分别有解为：$b^{\frac{p+1}{4}}$和 $p-b^{\frac{p+1}{4}}$；$b^{\frac{q+1}{4}}$和 $q-b^{\frac{q+1}{4}}$。现在求 $x^2=b \bmod n$ 的解，必须是两个方程都满足的共同解。其次，应将解得区域[1，$p-1$]和[1，$q-1$]扩展到[1，n]，为此，第一个方程的解系可由 $b^{\frac{p+1}{4}}$和 $p-b^{\frac{p+1}{4}}$加上 p 的整数倍得到，第二个方程的解系可由 $b^{\frac{q+1}{4}}$和 $q-b^{\frac{q+1}{4}}$加上 q 的整数倍得到。最后，从两个解系中找到共同的四个解。

【例 2】 $p=3$，$q=7$，$n=21$，求解 $x^2=1 \bmod 21$。

解：因为$\frac{p-1}{2}$，$\frac{q-1}{2}$都是奇数，所以：

① 显然 $b=1$ 满足 $b^{\frac{p-1}{2}}=1$ 和 $b^{\frac{q-1}{2}}=1$，故知 $b=1$ 是平方剩余。故：

$x_1=b^{\frac{p+1}{4}}=b=1$ 和 $p-1=2$ 是 $x^2=1 \bmod p$ 的两个解；

$x_2=b^{\frac{q+1}{4}}=b^2=1$ 和 $q-1=6$ 是 $x^2=1 \bmod q$ 的两个解；

第一个方程解系为 {1，2，1+3，2+3，1+6，2+6，…}；

第二个方程解系为 {1，6，1+7，6+7，1+14，6+14，…}；

共同解为 $\alpha_1=1$ 和 $\alpha_2=8$，另外两个是 $21-\alpha_1=20$ 和 $21-\alpha_2=13$。

② 还可验证 $b=4$ 也是平方剩余。这是因为 $b^{\frac{p-1}{2}}=4^1 \bmod p=1$ 和 $b^{\frac{q-1}{2}}=4^3 \bmod q=64 \bmod 7=1$。故：

$x_1=b^{\frac{p+1}{4}}=b^1=4 \bmod 3=1$ 和 $p-x_1=3-1=2$ 是方程 $x^2=4 \bmod 3$ 的两个解；

$x_2=b^{\frac{q+1}{4}}=b^2=16 \bmod 7=2$ 和 $q-x_2=7-2=5$ 是方程 $x^2=4 \bmod 7$ 的两个解；

扩展后的解系为 {1，2，4，5，7，8，…}和{2，5，9，12，…}

共同解为 $\alpha_1=2$ 和 $\alpha_2=5$，另两个是 $21-2=19$ 和 $21-5=16$。

③ 还可以验证 $b=16$ 是平方剩余。这是因为 $16^{\frac{p-1}{2}}=16 \bmod p=1$ 和 $16^{\frac{q-1}{2}}=16^3 \bmod q=2^3 \bmod q=1$。故：

$x_1=b^{\frac{p+1}{4}}=16 \bmod p=1$ 和 $p-x=2$ 是方程 $x^2=16 \bmod 3$ 的两个解；

$x_2=b^{\frac{q+1}{4}}=16^2 \bmod q=4 \bmod 7$ 和 $q-4=3$ 是方程 $x^2=16 \bmod 7$ 的两个解；

扩展后的解系为{1，2，4，5，7，8，10，11，…}和{3，4，10，11，…}

共同解为 $\alpha_1=4$，$\alpha_2=10$，另外两个是 $21-4=17$ 和 $21-10=11$，共有$\frac{1}{4}(p-1)(q-1)=3$ 个 QR。

(2) $\frac{p-1}{2}$或$\frac{q-1}{2}$为偶数的情况。

这时，不能直接代公式写出它的解 $b^{\frac{p+1}{4}}$或 $b^{\frac{q+1}{4}}$，然而如果用其他方法，比如穷举试解法(或用计算机程序搜索)得到了一个解，则求另一个解和共同解的做法同上。

【例 3】 $p=13$，$q=5$，$n=65$，求解 $x^2=1 \bmod 65$。

解：这时$\frac{p-1}{2}=6$，$\frac{q-1}{2}=2$，均为偶数，因此它共有$\frac{1}{4}(p-1)(q-1)=12$ 个 QR，不难验证 $b=1$ 是 p 和 q 的平方剩余。

对于 $b=1$，有 $b^{\frac{p-1}{2}}=b^1=1$ 和 $b^{\frac{q-1}{2}}=b^2=1$，因此有：$x^2=1 \bmod p$ 的解是 $x=1$ 和 $x=p-1=12$；$x^2=1 \bmod q$ 的解是 $x=1$ 和 $x=q-1=4$。

从而，解系为{1，12，14，25，…}和{1，4，6，9，11，14，…}；共同解为 $x=1$，$x=14$，以及 $65-1=64$ 和 $65-14=51$。

4.4.2 Rabin 密码的安全性

先来证明，求解一个属于 QR 元素的平方根的计算，等价于分解 n 为因子的计算。如果能分解 $n=pq$，则解 $x^2=a \bmod n$ 等价于解 $x^2=a \bmod p$ 和 $x^2=a \bmod q$。设它们的解分别是 $\pm x_1$ 和 $\pm x_2$，且 $|x_1| \neq |x_2|$，则有：

$$x_1^2 = x_2^2 \bmod n \tag{4-20}$$

即

$$(x_1+x_2)(x_1-x_2) = 0 \bmod n \tag{4-21}$$

它等价于 $p|(x_1+x_2)$ 而 $q|(x_1-x_2)$，或者 $p|(x_1-x_2)$ 而 $q|(x_1+x_2)$。其中，p 和 q 不能同时整除 (x_1+x_2) 和 (x_1-x_2)。

求出了 x_1 和 x_2 就等于求出了 p 和 q。表明二次剩余的复杂度与分解大数相同，也就是说 Rabin 的复杂度不低于 RSA，它除了求两平方根问题外，还要求解中国剩余问题（二方程联立求平方剩余）。

4.5 ElGamal 公钥系统（基于离散对数）

ElGamal 公钥系统是基于 GF(p)域中离散对数的 Np 复杂性而设计的。

4.5.1 基本算法

设 p 是素数，g 是 Z_p^* 中的本原元素，选取 $\alpha \in [0, p-1]$，计算：

$$\beta = g^{\alpha} \bmod p \tag{4-22}$$

设私有密钥为 $k_2=\alpha$，公有密钥为 $k_1=(g, \beta, p)$，则对于待加密消息 m，选取随机数 $k \in [0, p-1]$，加密算法为

$$E_{k_1}(m, k) = (y_1, y_2) \tag{4-23}$$

其中：

$$y_1 = g^k \bmod p, \quad y_2 = m\beta^k \bmod p \tag{4-23'}$$

解密算法为

$$D_{k_2}(y_1, y_2) = y_2(y_1^{\alpha})^{-1} \bmod p \tag{4-24}$$

这是因为

$$y_2(y_1^{\alpha})^{-1} = m\beta^k \cdot g^{-\alpha k} = mg^{\alpha k} \cdot g^{-\alpha k} = m \bmod p \tag{4-24'}$$

算法中离散对数的复杂性决定了系统的安全，然而，算法的设计是基于 Z_p^* 群是 p 为模的同余乘法群，对乘法满足封闭性，且存在乘法模逆元。如果 p 不是素数，则（模 p）同余类群 Z_p 只能是以加法为运算的整数群，只存在加法模逆元，只对加法满足封闭性，离散对

数的关系就不存在了。

4.5.2 有限群上的离散对数公钥体系

将 Z_p^* 群推广到一般的有限群 G，便得到推广的 ElGamal 公钥体制。

设群 G 是运算符为 $*$ 的有限群，$\alpha \in G$，H 是由 α 生成的 G 的子群，$H=\{\alpha^i \mid i \geqslant 0\}$，选取 $a \in H$，计算 $\beta=\alpha^a$，则私有密钥为 a，公有密钥为 α、β。

对于明文 m，选择随机数 $k \in [0, |H|-1]$，则加密算法为

$$E_{k_1}(m, k) = (y_1, y_2) \tag{4-25}$$

其中：

$$y_1 = \alpha^k, \quad y_2 = m\beta^k \tag{4-26}$$

解密算法为

$$D_{k_2}(y_1, y_2) = y_2 \cdot (y_1^a)^{-1} \tag{4-27}$$

这是因为：

$$y_2 \cdot (y_1^a)^{-1} = m\beta^k \alpha^{-ak} = m\alpha^{ak}\alpha^{-ak} = m$$

4.5.3 离散对数计算法

离散对数公钥体系的安全性是基于计算离散对数的复杂性的。无论从破译(分析)这种公钥体系的还是从改进(提高)这种公钥体系的安全性出发，都应了解计算离散对数的方法。

离散对数是模幂运算的逆运算。计算模幂 a^x 并不困难。将 x 表达为二进制形式：

$$x = b_0 + b_1 \cdot 2 + b_2 \cdot 2^2 + \cdots + b_{n-1} \cdot 2^{n-1} \tag{4-28}$$

则

$$a^x = a^{b_0} \times (a^2)^{b_1} \times ((a^2)^2)^{b_2} \times \cdots \times (a^{2^{n-1}})^{b_{n-1}} \bmod p$$

式中，$b_i=0$ 的因子代之为 1，$b_i=1$ 的因子都化为 a 的各级二次幂。如求 $5^{375} \bmod 1823$，则因为

$$375 = (101110111)_2 = 1 + 2 + 2^2 + 2^4 + 2^5 + 2^6 + 2^8$$

所以

$$5^{375} = 5 \cdot 5^2 \cdot 5^4 \cdot 5^{16} \cdot 5^{32} \cdot 5^{64} \cdot 5^{256}$$

求出 $5=5$，$5^2=25$，$5^4=625$ 后，继续求得：

$$5^8 = 625^2 \bmod 1823 = 503, \quad 5^{16} = 503^2 \bmod 1823 = 1435$$

$$5^{32} = 1435^2 \bmod 1823 = 1058, \quad 5^{64} = 1058^2 \bmod 1823 = 42$$

$$5^{128} = 42^2 \bmod 1823 = 1764, \quad 5^{256} = 1764^2 \bmod 1823 = 1658$$

所以

$$5^{375} = 5 \times 25 \times 625 \times 1435 \times 1058 \times 42 \times 1658 \bmod 1823 = 591$$

然而要求出 $\log_5 591 = ? \bmod 1823$ 就不容易了。

已知 $a^x = y \bmod p$，求 $\log_a y = x \bmod p$ 的运算就是离散对数。

1. 商克(Shanks)法

先看一例子：在 GF(23)**上求** $\log_5 3$。**因为数值较小，我们把** $x=[1, 22]$**对应的全部** $y=5^x \bmod 23$ **都计算出来，列于表** 4.1 **中。为了查对数“**$x=\log_5 y \bmod 23$**”的方便，数值已按** y **排序。查表得到** $\log_5 3=16$。

表 4.1 $y=5^x \bmod 23$ 的全部整数解

y	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
x	22	2	16	4	1	18	19	6	10	3	9	20	14	21	17	8	7	12	15	5	13	11

这种解法实际是穷举法，p **很大时复杂度将很高。**

由表可以看到，$5^{22} \bmod 23=5^0 \bmod 23=1 \bmod 23$**，这正是费尔马定理** $a^{p-1}=1 \bmod p$。**它表明** $n=22$ **是** x **的周期，**5 **自乘** 22 **次就回到了** 1。**同时表明原方程解的取值范围是** $0 \leqslant x<n$。

为了减少搜索范围，现在把 n **分成了** d **段，每段** d **个值，这里：**

$$d=[\sqrt{n}]=[\sqrt{22}]=5$$

首先只在 $0 \leqslant r<d$ **内搜索满足**

$$a^r=b \bmod p$$

的那些 r **值，它们被列在表** 4.2 **中。对于大于** d **的那些解** x**，可令：**

$$x=qd+r \tag{4-29}$$

再进行 q **轮的搜索，而** q **是整商，由于** $x \leqslant n$，$q=\dfrac{x-r}{d} \leqslant \dfrac{n-r}{d} \approx \sqrt{n}$**(但不大于**$\sqrt{n}$**)，所以** $0 \leqslant q<d$。**搜索轮数也不会大于** d。**这时：**

$$y=a^x=a^{qd+r} \tag{4-30}$$

利用 $a^n=1 \bmod p$**(**n **为周期)，就有：**

$$a^r=a^{qd+r} \cdot a^{-qd}=y \cdot (a^{-d})^q=y \cdot (a^{n-d})^q \bmod p \tag{4-31}$$

离散对数问题中，a **和** y **是已知的，**n **与** d **也都容易得知，利用上式，分别令** $q=0$，1，2，3，**…去测试，而每次测试只是在表** 4.2 **的** d **个数据内搜索，总共最多搜索** d **轮。**

表 4.2 0～d 范围 $r=a^r \bmod 23$ 的解

$b=5^r$	1	2	4	5	10
r	0	2	4	1	3

【例 4】 已知 $y=3$，$a=5$，**求满足** $y=a^x \bmod 23$ **的** x。**此例主要用以说明商克法的使用方法。**

解：根据周期 $n=22$，**取** $d=5$；**赋初值** $a=5$，$y=3$。**计算** $a^{n-d}=5^{22-5}=5^{17} \bmod 23=15$，**存入** A，**且令** $q \leftarrow 0$。

第一轮，计算

$$b=a^r=y(a^{n-d})^0=3 \times 15^0=3$$

表中未查到，将结果 b **存入** B，**令** $q \leftarrow q+1$，**这时** $q=1$；

第二轮，计算

$$b = y \cdot (a^{n-d})^1 = y \cdot (a^{n-d})^0 \cdot (a^{n-d}) = B \times A = 3 \times 15 \bmod 23 = 22$$

表中仍未查到，将结果 b 存入 B，令 $q \leftarrow q+1$，这时 $q=2$；

第三轮，计算

$$b = y \cdot (a^{n-d})^2 = B \times A = 22 \times 15 \bmod 23 = 8$$

表中还是未查到，将 b 存入 B，令 $q \leftarrow q+1$，这时 $q=3$；

第四轮，计算

$$b = y \cdot (a^{n-d})^3 = B \times A = 8 \times 15 \bmod 23 = 5$$

表中查到了，是 $r=1$，这时 $q=3$。因此，结果是：

$$x = qd + r = 3 \times 5 + 1 = 16$$

总结以上算法，得到程序算法描述如下：

$S1$：选择 $d \approx \sqrt{n}$，$r \leftarrow 0$，$b \leftarrow 1$。

$S2$：进入表格$(b \sim r)$，查表。

$S3$：若 $r=d-1$，则作：对表格中 b 进行排序，转 $S4$。

否则作：始 $b \leftarrow ab \bmod p$，$r \leftarrow r+1$，转 $S2$。结束。

$S4$：计算 $A \leftarrow a^{n-d}$，$B \leftarrow y$，$q \leftarrow 0$，开始下一轮。

$S5$：若查表存在$(b \sim r)$，其中 $b=B$，则作：$x \leftarrow qd+r$ 输出 x，结束。

否则作：$B \leftarrow BA$，$q \leftarrow q+1$，转 $S5$。

2. 波里格－黑尔曼(Pohlig -Hellman)算法

此算法适应于

$$p - 1 = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k} \qquad (n_i > 0,\ 1 \leqslant i \leqslant k) \tag{4-32}$$

的情况，即 $p-1$ 可分解成许多小素数因子的情况。

为了求解：

$$y = x^r \bmod p \tag{4-33}$$

可令

$$r = r_i \bmod p_i^{n_i} \qquad (1 \leqslant i \leqslant k) \tag{4-34}$$

根据中国剩余定理，只要能确定各个 r_i，原方程的解 r 即可知晓：

$$r = r_1 M_1 y_1 + r_2 M_2 y_2 + \cdots + r_k M_k y_k \tag{4-35}$$

式中：

$$M_i = \frac{p-1}{p_i^{n_i}},\ M_i y_i = 1 \bmod p_i^{n_i}$$

而 r_i 可设为

$$r_i = r_{i0} + r_{i1} p_i + r_{i2} p_i^2 + \cdots + r_{in_i-1} p_i^{n_i-1}$$

先来讨论 $r=r_1 \bmod p_1^{n_1}$。将式(4-35)代入式(4-33)，经过一些推导：

① 可得到

$$y^{(p-1)/p_1} = x^{r(p-1)/p_1} \bmod p = x^{r_1(p-1)/p_1} \bmod p = x^{r_{10}(p-1)/p_1} \bmod p$$

由于离散对数问题中 x 和 y 都是已知的，因此比较等号两边就可以确定 r_{10}；

② 令 $y_1 = yx^{-r_{10}}$，还可得到

$$y_1^{(p-1)/p_1^2} = x^{r_{11}(p-1)/p_1} \bmod p$$

由此可确定 r_{11}。

③ 令 $y_2=y_1x^{-r_{11}p_1}$，则有

$$y_2^{(p-1)/p_1^3}\equiv x^{r_{12}(p-1)/p_1} \bmod p$$

由此可确定 r_{12}。

类似地，令 $y_3=y_2x^{-r_{12}p_1^2}$，则有

$$y_3^{(p-1)/p_1^4}\equiv x^{r_{13}(p-1)/p_1} \bmod p$$

……

由此可确定 r_{13}、r_{14}…… 从而：

$$r_1=r_{10}+r_{11}p_1+r_{12}p_1^2+\cdots+r_{1(n_i-1)}p_1^{(n_i-1)}$$

对各 $p_i^{n_i}$ 的子方程同法处理，最后由式(4-35)可写出 r。

【例5】 $p=8101$，$x=6$，$y=7833$，求满足 $y\equiv x^r \pmod p$ 的 r。

解：首先由 $p-1=8100=2^2\times3^4\times5^2$ 知 $p_1=2$，$p_2=3$，$p_3=5$。令 $\beta_i=x^{(p-1)/p_i}$。

(1) 对 $p_1=2$，有：

$$\beta_1=6^{8100/2}=6^{4050}=8100 \pmod{8101},\ \beta_1^2=1$$

另一方面，$p_1=2$，$n_1=2$，$y=7833$，$y^{8100/2}=8100 \pmod{8101}$，代入①可求得 $r_{10}=1$。又 $x^{-1}=6751$，$y_1=yx^{-1}=5356$，$y_1^{8100/4}=1$，代入②可求得 $r_{11}=0$。所以

$$r_1=r_{10}+r_{11}p_1=1$$

(2) 对 $p_2=3$，有：

$$\beta_2=6^{8100/3}=6^{2700}=5883 \pmod{8101},\ \beta_2^2=2217,\ \beta_2^3=1$$

另一方面，$p_2=3$，$n_2=4$，$y=7833$，$y^{8100/3}=2217$，代入①可求得 $r_{20}=2$。

同法求得 $r_{21}=2$，$r_{22}=1$，$r_{23}=1$，所以

$$r_2=2+2\times3+3^2+3^3=44$$

(3) 对于 $p_3=5$，有：

$$\beta_3=6^{8100/5}=6^{1620}=3547,\ \beta_3^2=356,\ \beta_3^3=7077,\ \beta_3^4=5221,\ \beta_3^5=1$$

另一方面，$p_3=5$，$n_3=2$，$y^{8100/5}=356$，代入①可求得 $r_{30}=2$。又 $y_1=yx^{-2}=7833\times7876=3593$，$y_1^{8100/25}=356$，代入②可求得 $r_{31}=2$。所以

$$r_3=2+2\times5=12$$

(4) 现在就可以用中国剩余定理来求 r 了(还利用了 $a^{-1}=a^{\phi(m)-1} \bmod m$)：

$M_1=3^45^2=2025$，$y_1=M_1^{-1} \bmod 2^2=2025^1 \bmod 4=1$，$M_1y_1=2025 \pmod{8101}$

$M_2=2^25^2=100$，$y_2=M_2^{-1} \bmod 3^4=100^{53} \bmod 81=64$，$M_2y_2=6400 \pmod{8101}$

$M_3=2^23^4=324$，$y_3=M_3^{-1} \bmod 5^2=324^{15} \bmod 25=24$，$M_3y_3=7776 \pmod{8101}$

所以

$$\begin{aligned}r&=r_1M_1y_1+r_2M_2y_2+r_3M_3y_3\\&=2025\times1+44\times6400+12\times7776\\&=376\ 937=4337\ (\bmod 8101)\end{aligned}$$

以上求解过程表明，在 GF(p)域中使用离散对数密码体制时，$p-1$ 有小因数是不安全的。最好 $p-1=2p_1$，而 p_1 是大素数。而在 GF(2^m)域中，则希望 2^m-1 是大素数，如 $m=127$ 时，$N=2^{127}-1$ 是大素数，$\sqrt{N}=10^{19}$；当 $m=521$ 时，$N=2^{521}-1$ 也是大素数，$\sqrt{N}=10^{78}$。

4.6 *McEliece* 公钥密码(基于纠错码)[①]

纠错码和密码是编码的不同分支，在 20 世纪 70 年代之前它们几乎互不相关，各自独立发展。随着现代密码学的诞生，计算复杂性成了密码体系安全的基础；而随着纠错码的发展，许多代数方法的引入，纠错码中也出现了许多 NPC 问题，如 1997 年证明了一般线性码的 Hamming 重量问题是 NPC 问题，还有：

(1) 陪集重量问题是 NPC 问题。

(2) 重量分布问题是 NPC 问题。

(3) 最小码距问题是 NPC 问题。

如此等等。1978 年，McEliece 基于纠错码设计出了公钥体制，它利用了郭帕(Goppa)码具有快速译码的特点，发明了 McEliece 码，此后纠错码成了构造公钥密码的又一热点。

4.6.1 Goppa 码的一般描述[12]

1. Goppa 码的有理分式表示

正如可以利用一个多项式 $C(x)=c_{n-1}x^{n-1}+c_{n-2}x^{n-2}+\cdots+c_1x+c_0$ 表示一个 n 维线性空间的矢量$(c_{n-1}c_{n-2}\cdots c_2c_1c_0)$一样，也可以用一个有理分式来表示这个矢量：

$$C(z)=\frac{c_{n-1}}{z-\alpha_{n-1}}+\frac{c_{n-2}}{z-\alpha_{n-2}}+\cdots+\frac{c_1}{z-\alpha_1}+\frac{c_0}{z-\alpha_0}=\sum_{i=0}^{n-1}\frac{c_i}{z-\alpha_i} \tag{4-36}$$

如果 $C(x)$是一个(n,k)循环码的码多项式，则应满足：

$$C(x)=0 \bmod g(x)$$

其中，$g(x)$是生成多项式。

同理，如果要把 $C(z)$作为循环码的有理多项式，它也应当被生成多项式 $g(z)$整除：

$$C(z)=\sum_{i=0}^{n-1}\frac{c_i}{z-\alpha_i}\equiv 0\ (\bmod\ g(z)) \tag{4-37}$$

显然，若

$$a(z)=\sum_{i=0}^{n-1}\frac{a_i}{z-\alpha_i}\equiv 0,\ b(z)=\sum_{i=0}^{n-1}\frac{b_i}{z-\alpha_i}\equiv 0 \bmod g(z) \tag{4-38}$$

是某个码组中的两个码字，则不难证明：

$$a(z)+b(z)=\sum_{i=0}^{n-1}\frac{a_i+b_i}{z-\alpha_i}=\sum_{i=0}^{n-1}\frac{c_i}{z-\alpha_i}\equiv 0 \bmod g(z) \tag{4-39}$$

也是一个码字。可见有理分式码是线性码。

下面讨论有理分式表示与多项式表示的关系。

由于生成多项式 $g(z)$是既约多项式，因此 $z-\alpha_i$ 必与它互素，所以 $z-\alpha_i$ 必存在逆元

$$\frac{1}{z-\alpha_i}=p_i(z) \bmod g(z)$$

其中，$p(z)$是次数不高于 $g(z)$的多项式。于是：

---

① 本节为扩展知识，可不讲或只作简单介绍。

$$\sum_{i=0}^{n-1}\frac{c_i}{z-\alpha_i}=\sum_{i=0}^{n-1}c_i p_i(z)\equiv 0 \bmod g(z) \tag{4-40}$$

如果有理分式各多项式都采用模逆元，就变成了多项式表达。

2. Goppa 码的定义和描述

设 $0<n\leqslant q^m$（q 为素数或素数幂，m 为大于 0 的整数），$L=\{\alpha_1\alpha_2\cdots\alpha_n\}$ 是一个有序集合，$\alpha_i\in \mathrm{GF}(q^m)$，且对任何不大于 n 的正整数 i 和 j，当 $i\neq j$ 时恒有 $\alpha_i\neq\alpha_j(i,j\leqslant n)$；又设 $\mathrm{GF}(q)$ 上 n 维线性空间为 V_n，矢量 $\boldsymbol{C}=(c_1c_2\cdots c_n)\in V_n$，则 $\boldsymbol{C}$ 对应的 $\mathrm{GF}(q^m)$ 上 z 的有理分式表示为

$$R_c(z)=\sum_{i=1}^{n}\frac{c_i}{z-\alpha_i} \tag{4-41}$$

再设 $g(z)$ 是系数在 $\mathrm{GF}(q^m)$ 上的 z 的多项式，它的根不在 L 中，则以下 n 重矢量集合：

$$\{\boldsymbol{C}\mid R_c(z)\equiv 0 \bmod g(z),\ \boldsymbol{C}\in V_n\} \tag{4-42}$$

被称为由 $g(z)$ 生成的 Goppa 码，而 $g(z)$ 则称为生成多项式或 Goppa 多项式。若 $g(z)$ 在 $\mathrm{GF}(q^m)$ 上既约，则称为既约 Goppa 码。

显然，Goppa 码是一个线性码。

【例 6】 已知 $C=(11110001)$，给出它的一个有理分式表达。

解：因 $n=8$，取 $L=\{\alpha_1\alpha_2\cdots\alpha_8\}$，$\alpha_i\in\mathrm{GF}(2^3)$；取 $g(z)=z^2+z+1$，则 $C=(11110001)$ 的有理分式表示为

$$R_c(z)=\frac{1}{z-\alpha_1}+\frac{1}{z-\alpha_2}+\frac{1}{z-\alpha_3}+\frac{1}{z-\alpha_4}+\frac{1}{z-\alpha_8}=\frac{f'(z)}{f(z)}$$

式中：

$$f(z)=(z-\alpha_1)(z-\alpha_2)(z-\alpha_3)(z-\alpha_4)(z-\alpha_8)$$

$$f'(z)=\frac{\mathrm{d}}{\mathrm{d}z}f(z)=\sum_j\prod_{j\neq i}(z-\alpha_i)$$

为了验证 $R_c(z)\equiv 0 \bmod g(z)$ 是否成立，只要检查 $f'(z)\equiv 0 \bmod g(x)$ 成立即可。能整除则是码字，否则不是。

决定一个 Goppa 码的两个参数，一个是有序集合 L，另一个是生成多项式 $g(z)$，因此又将 Goppa 码写为 $\Gamma(L,g)$ 码。由于 $f(z)=\prod_i(z-\alpha_i)$，表明 $f(z)$ 的根在 L 之内，同时定义指出 $g(z)$ 的根不在 L 之内，因而 $(f(z),g(z))=1$ 互素，这是对 $g(z)$ 的基本要求。

再若 $g(z)$ 和 $f(z)$ 的根都在 $\mathrm{GF}(q^m)$ 域内，那么码长

$$n=q^m-\partial^\circ g(z)$$

式中，$\partial^\circ g(z)$ 表示 $g(z)$ 的次数。

如果 $g(z)$ 的根不在 $\mathrm{GF}(q^m)$ 之中，而在 $\{L\}$ 的扩域 $\{\alpha_1\alpha_2\cdots\alpha_{q^m}\}$ 中，于是码长可取 $n=q^m$。

3. Goppa 码的校验矩阵和最小码距

如果改写 $R_c(z)=\sum_{i=0}^{n-1}\frac{c_i}{z-\alpha_i}\equiv 0 \bmod g(z)$ 为矩阵形式：

$$\left(\frac{1}{z-\alpha_0}\cdot\frac{1}{z-\alpha_1}\cdots\frac{1}{z-\alpha_{n-1}}\right)\cdot\begin{pmatrix}c_0\\c_1\\\vdots\\c_{n-1}\end{pmatrix}=\boldsymbol{H}_1\boldsymbol{C}^{\mathrm{T}}\equiv 0\ (\bmod\ g(x)) \tag{4-43}$$

则称 $\boldsymbol{H}_1$ 为 Goppa 码的校验矩阵。$\boldsymbol{H}_1$ 还可以改变形式。由

$$\frac{g(z)}{z-\alpha_i} \equiv 0 \bmod g(z) \tag{4-44}$$

可写

$$\frac{-1}{z-\alpha_i} = \frac{g(z)-g(\alpha_i)}{z-\alpha_i} \cdot g^{-1}(\alpha_i) \pmod{g(z)} \tag{4-45}$$

把 $\boldsymbol{H}_1$ 的每一个矩阵元都用上式代入，得：

$$\boldsymbol{H}_2 = -\left(\frac{g(z)-g(\alpha_0)}{z-\alpha_0}g^{-1}(\alpha_0),\ \frac{g(z)-g(\alpha_1)}{z-\alpha_1}g^{-1}(\alpha_1),\ \cdots,\ \frac{g(z)-g(\alpha_{n-1})}{z-\alpha_{n-1}}g^{-1}(\alpha_{n-1})\right) \tag{4-46}$$

负号并不影响校验，即

$$\boldsymbol{H}_2\boldsymbol{C}^{\mathrm{T}} = 0 \tag{4-47}$$

经化简，$\boldsymbol{H}_2$ 可写成以下形式：

$$\boldsymbol{H}_2 = \begin{pmatrix} g_r & 0 & \cdots & 0 \\ g_{r-1} & g_r & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ g_1 & r_2 & \cdots & r_r \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \alpha_0^2 & \alpha_1^2 & \cdots & \alpha_{n-1}^2 \\ \vdots & \vdots & & \vdots \\ \alpha_0^{r-1} & \alpha_1^{r-1} & \cdots & \alpha_{n-1}^{r-1} \end{pmatrix} \cdot \begin{pmatrix} g^{-1}(\alpha_0) & & & \\ & g^{-1}(\alpha_1) & & \\ & & \ddots & \\ & & & g^{-1}(\alpha_{n-1}) \end{pmatrix}$$

$$= \boldsymbol{A}\boldsymbol{\alpha}\boldsymbol{B} \tag{4-48}$$

$\boldsymbol{A}$ 矩阵最后可化为只有主对角线的矩阵，它不影响校验和纠错能力，对校验有意义的是 $\boldsymbol{H}_3 = \boldsymbol{\alpha}\boldsymbol{B}$，即

$$\boldsymbol{H}_3 = \begin{pmatrix} g^{-1}(\alpha_0) & g^{-1}(\alpha_1) & \cdots & g^{-1}(\alpha_{n-1}) \\ \alpha_0 g^{-1}(\alpha_0) & \alpha_1 g^{-1}(\alpha_1) & \cdots & \alpha_{n-1} g^{-1}(\alpha_{n-1}) \\ & & \vdots & \\ \alpha_0^{r-1} g^{-1}(\alpha_0) & \alpha_1^{r-1} g^{-1}(\alpha_1) & \cdots & \alpha_{n-1}^{r-1} g^{-1}(\alpha_{n-1}) \end{pmatrix} \tag{4-49}$$

由于 $\boldsymbol{\alpha}$ 矩阵的秩为 r，$\boldsymbol{B}$ 为满秩，因此 $\boldsymbol{H}_3$ 的秩为 r，所以不仅 r 行线性无关，而且任意 r 列线性无关。因此，Goppa 码的最小码距为

$$d = r+1 = \partial^{\circ} g(z)+1 \qquad (g(r) \text{ 的次数为 } r) \tag{4-50}$$

定理(结论)：由 r 次多项式 $g(z)$生成的 q 进制 Goppa 码，至少有 mr 个校验位，最小码距 $d \geqslant r+1$。如果 $g(z)$的根不在 GF(q^m)内，则能生成码长 $n=q^m$，信息位长 $k \geqslant q^m - mr$ 的 Goppa 码。

理论上，当 $n \to \infty$时，一定有达到 Shannon 限的 Goppa 码存在，但实际上这个 $g(z)$如何取却没有得到答案。

4. 二进制 Goppa 码

二进制下 $C=(c_0 c_1 \cdots c_{n-1})$，各 c_i 非 0 即 1，可写为

$$R_c(z) = \frac{f_1(z)}{f(z)} = \varepsilon(z) \tag{4-51}$$

式中：

$$f(z) = f_0 + f_1 z + f_2 z^2 + f_3 z^3 + \cdots \tag{4-52}$$

$$f_1(z) = f'(z) = f_1 + 2f_2 z + 3f_3 z^2 + 4f_4 z^3 + \cdots = f_1 + f_3 z^2 + \cdots \pmod 2 \tag{4-53}$$

是一个偶函数。

设 $\overline{g}(z)$是除尽 $g(z)$的次数最低的完全偶多项式 $\overline{g}(z) \mid g(z)$，则有

$$\varepsilon(z) \equiv 0 \bmod \overline{g}(z) \tag{4-54}$$

若 $g(z)$无重根，则 $\overline{g}(z)=g^2(z)$，故二进制 Goppa 码的最小码距为

$$d \geqslant \partial^\circ \overline{g}(z) + 1 = 2\partial^\circ g(z) + 1 \tag{4-55}$$

信息位 $k=n-\partial^\circ g(z)m$。

【例 7】 已知二进制 Goppa 码的生成多项式为 $g(z)=z^2+z+1$，求编码后的码字。

解：$L=\mathrm{GF}(2^3)=\{0, 1, \alpha, \alpha^2, \cdots, \alpha^6\}$，$\alpha$ 是本原元素，本原多项式是 x^3+x+1。

当 $i=0, 1, 2, \cdots, 6$ 时，$g(\alpha_0^i)\neq 0$，表明 $g(z)$在 $\mathrm{GF}(2^3)$既约，所以 $g(z)$的根必在 $\mathrm{GF}(2^3)$、$\mathrm{GF}(2^4)$、$\mathrm{GF}(2^6)$等域中。由此得到一个既约 Goppa 码：

$$n = 2^3 = 8,\ k \geqslant 8 - 3 \times 2,\ d \geqslant 5$$

校验矩阵为 $k=2$ 行，$n=8$ 列的矩阵：

$$\boldsymbol{H} = \begin{pmatrix} g^{-1}(0) & g^{-1}(1) & g^{-1}(\alpha) & g^{-1}(\alpha^2) & \cdots & g^{-1}(\alpha^6) \\ 0 \cdot g^{-1}(0) & 1 \cdot g^{-1}(1) & \alpha \cdot g^{-1}(\alpha) & \cdots & \cdots & \alpha^6 \cdot g^{-1}(\alpha^6) \end{pmatrix}$$

在 $\mathrm{GF}(2^3)$域中，本原元素 α^i 应满足 $P(\alpha)=\alpha^3+\alpha+1=0$ 的本原多项式的约束，即满足模 $P(\alpha)$的运算法则，如表 4.3 所示。

表 4.3 GF(2^3)域中，本原元素 α^i 的模 $P(\alpha)$的运算法则

α^i	$\alpha^0=1$	$\alpha^1=\alpha$	α^2	α^3	α^4	α^5	α^6
等价为	1	α	α^2	$\alpha+1$	$\alpha^2+\alpha$	$\alpha^2+\alpha+1$	α^2+1
二元表示	001	010	100	011	110	111	101

α^i 还应满足 $q=7$ 的循环乘法 $\alpha^7=\alpha^0=1$，因此 α^k 与 α^{7-k} 互为模逆元。比如 $g(\alpha)=\alpha^2+\alpha+1=\alpha^5$，则 $g^{-1}(\alpha)=\alpha^2$；又如 $g(\alpha^2)=\alpha^4+\alpha^2+1=(\alpha^2+\alpha)+\alpha^2+1=\alpha+1=\alpha^3$；则 $g^{-1}(\alpha^2)=\alpha^4=\alpha^2+\alpha$。同理，所有的 $g(\alpha^i)$ 和 $g^{-1}(\alpha^i)$都可算出，列于表 4.4 中。

表 4.4 $g(\alpha^i)$和 $g^{-1}(\alpha^i)$的运算结果

α^i	0	1	α	α^2	α^3	α^4	α^5	α^6
$g(\alpha^i)$	1	1	α^5	α^3	α^5	α^6	α^6	α^3
$g^{-1}(\alpha^i)$	1	1	α^2	α^4	α^2	α	α	α^4

根据这些数据，$\boldsymbol{H}$ 矩阵就可以写为

$$\boldsymbol{H} = \begin{pmatrix} 1 & 1 & \alpha^2 & \alpha^4 & \alpha^2 & \alpha & \alpha & \alpha^4 \\ 0 & 1 & \alpha^3 & \alpha^6 & \alpha^5 & \alpha^5 & \alpha^6 & \alpha^3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

由方程 $\boldsymbol{H}\cdot\boldsymbol{C}^{\mathrm{T}}=0$ 可解得四个码字(或者说以下四个码字均能满足 $\boldsymbol{H}\cdot\boldsymbol{C}^{\mathrm{T}}=0$),如表 4.5 所示。

表 4.5 由方程 $\boldsymbol{H}\cdot\boldsymbol{C}^{\mathrm{T}}=0$ 解出的四个码字

码字位置	0	1	α	α^2	α^3	α^4	α^5	α^6
码字 A	0	0	0	0	0	0	0	0
码字 B	0	0	1	1	1	1	1	1
码字 C	1	1	0	0	1	0	1	1
码字 D	1	1	1	1	0	1	0	0

从上面的四个码字中任选两个非 0 码字,进行初等变换后,即可得到系统码形式的生成矩阵,进而得到全部码字(本例太简单,只有 $k=2$,当然共四个码字)。

4.6.2 McEliece 密码体制

1. 系统参数

设 $\boldsymbol{G}$ 是 GF(2)上的$[n, k, d]$Goppa 码的生成矩阵,其中 $n=2^m$,$d=2i+1$,$k=n-mt$;设明文集合为 $\mathrm{GF}(2)^k$,密文集合为 $\mathrm{GF}(2)^n$(这里 k,n 表示位数)。随机选取 GF(2)上的 $k\times k$ 阶可逆矩阵 $\boldsymbol{S}$ 和 $n\times n$ 阶置换矩阵 $\boldsymbol{P}$,令

$$\boldsymbol{G}'=\boldsymbol{SGP} \tag{4-56}$$

则私有密钥为 $\boldsymbol{S}$、$\boldsymbol{G}$、$\boldsymbol{P}$,公开密钥为 $\boldsymbol{G}'$。

2. 加密算法

对于待加密明文 $\boldsymbol{m}\in\mathrm{GF}(2)^k$,其对应的密文为

$$\boldsymbol{C}=\boldsymbol{mG}'+\boldsymbol{z} \tag{4-57}$$

这里,z 是 $\mathrm{GF}(2)^n$ 上重量为 t 的随机向量(即 z 是长为 n,其中 t 个 1 的任意向量)。

3. 解密算法

收到 C 后,计算:

$$\boldsymbol{CP}^{-1}=\boldsymbol{mSGPP}^{-1}+\boldsymbol{zP}^{-1}=\boldsymbol{mSG}+\boldsymbol{z}' \tag{4-58}$$

由于 $\boldsymbol{P}$ 是置换矩阵,因此 z' 与 z 重量不变,即 $w(z')=w(z)=t$。视 z' 为 t 位错误格式矢量,$\boldsymbol{mSG}+z'$是发生了 t 位错的 $\boldsymbol{mSG}$,利用 Goppa 码的快速译码算法,经 Goppa 译码,错误得以纠正,将其译成 $\boldsymbol{m}'=\boldsymbol{mS}$,于是明文为

$$\boldsymbol{m}=\boldsymbol{m}'\boldsymbol{S}^{-1} \tag{4-59}$$

【例 8】 已知(7, 4, 3)码的生成矩阵为

$$\boldsymbol{G}=\begin{pmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{pmatrix}$$

根据 McELiece 密码体制设计加、解密算法。

解:假设用户 A 选择:

$$\boldsymbol{S}=\begin{pmatrix}1&1&1&0\\1&0&1&0\\1&1&0&1\\0&1&1&0\end{pmatrix},\quad \boldsymbol{P}=\begin{pmatrix}0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&1\\0&0&1&0&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&1&0\end{pmatrix}$$

则公开密钥为

$$\boldsymbol{G}'=\boldsymbol{SGP}=\begin{pmatrix}1&0&0&1&1&0&0\\1&0&1&1&0&1&0\\0&0&1&1&1&0&1\\1&1&1&0&1&0&0\end{pmatrix}$$

设明文 $m=(0101)$，错误格式向量 $z=(1000000)$，则密文

$$\boldsymbol{C}=\boldsymbol{mG}'+z=(1101110)$$

解密分两步，先作逆置换：

$$\boldsymbol{C}'=\boldsymbol{CP}^{-1}=\boldsymbol{CP}^{\mathrm{T}}=(1110011)=\boldsymbol{mSG}+z'$$

再经 Goppa 译码，得到：

$$\boldsymbol{m}'=\boldsymbol{mS}=(1100)$$

最后：

$$\boldsymbol{m}=\boldsymbol{m}'\boldsymbol{S}^{-1}=(1\ \ 1\ \ 0\ \ 0)\begin{pmatrix}1&0&0&1\\1&1&0&0\\1&1&0&1\\0&1&1&1\end{pmatrix}=(0\ \ 1\ \ 0\ \ 1)$$

因(7，3，4)码是汉明码，纠错能力为 1 位，所以任意一位错误的格式向量 z 都能解出相同的结果。

4.6.3 McEliece 的安全性

McEliece 最初的论文中建议码长为 1024，$t=50$，$k=524$，这是达到安全的基本要求。这时 $r=500$，S 为 500×500 的方阵，P 是 1024×1024 的置换方阵，而用 $t=50$ 的随机错误格式将数据打乱。

尽管 McEliece 是最早的公钥算法之一，该方案比 RSA 加解密速度快三个数量级，并且至今并未有对它攻击成功的分析结果，然而它却从未获得密码界的广泛接受。原因是它的公开密钥太庞大，为 10^{19} 比特长，而且密文是明文长度的两倍。

它的贡献在于开拓了基于纠错码的密码。

4.7 椭圆曲线公钥体制[13]

椭圆曲线理论是代数几何、数论等多个数学分支的定义，曾被认为是一个纯理论学科。20 世纪 80 年代，它被引入密码学后，由于它的复杂性高，密钥短，占用资源少，引起

了人们的极大兴趣，90 年代形成了研究热点，它在移动通信中的应用更加推动了对该领域研究的进展。

4.7.1 椭圆曲线

最初由椭圆弧长的积分引入，具有 $\int \frac{\mathrm{d}x}{\sqrt{E(x)}}$ 的形式。其中，$E(x)$是 x 的三次或四次多项式，积分无法解析表达，为此引入所谓的椭圆曲线函数，它是由以下的韦尔斯特拉斯(Weierstrass)方程确定的平面曲线：

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4-60}$$

这里，$a_i \in F(i=1, 2, \cdots, 6)$，$F$ 可以是有理数域，也可以是复数域，还可以是有限域。

将式(4-60)中的 y 代之以 $\frac{1}{2}(y-a_1x-a_3)$，方程可化为以 x 轴为对称轴的形式：

$$y^2 = 4x^3 + b_2x^2 + 2b_4x + b_6 \tag{4-61}$$

再以平移变换 $\left(\frac{x-3b_2}{36}, \frac{y}{216}\right)$ 代换两边的(x, y)，得到标准形式：

$$y^2 = x^3 + ax + b \tag{4-62}$$

下面对 $y=0$ 的根进行讨论。令

$$\Delta = 27b^2 + 4a^3 \tag{4-63}$$

根据 Δ 值的不同，方程的根有不同的取值，相应的椭圆曲线也有不同的形式：

(1) $\Delta>0$ 时，$y=0$ 有一个实根和一对复根。例如 $y^2=x^3-3x+3$(见图 4.1)和 $y^2=x^3+x$ (见图 4.2) 。

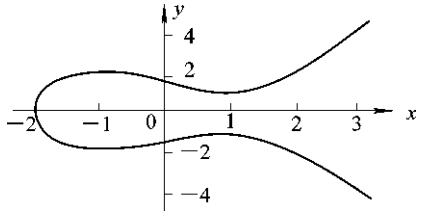

图 4.1 $y^2=x^3-3x+3$ 的曲线

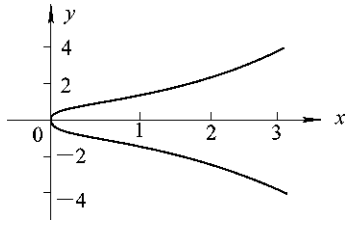

图 4.2 $y^2=x^3+x$ 的曲线

(2) $\Delta=0$ 时，$y=0$ 有三个实根，特别是 $\left(\frac{1}{2}b\right)^2=-\left(\frac{1}{3}a\right)^3$ 时，三个实根中有两个相等。例如 $y^2=x^3+x^2$(见图 4.3)。

(3) $\Delta<0$ 时，$y=0$ 有三个不等的实根。例如 $y^2=x^3-x$ (见图 4.4)。

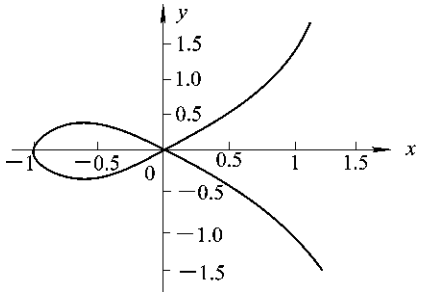

图 4.3 $y^2=x^3+x^2$ 的曲线

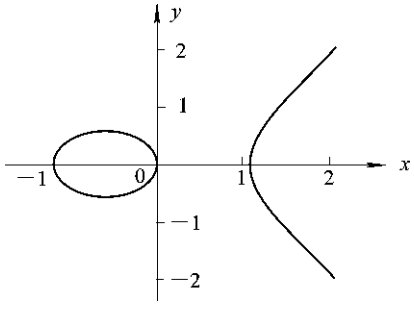

图 4.4 $y^2=x^3-x$ 的曲线

4.7.2 椭圆曲线上的点加运算

设 $P=(x_1,y_1)$，$Q=(x_2,y_2)$是椭圆曲线上的任意两个点。L 为连接 PQ 的直线，它交椭圆曲线于另一点 R，过 R 作平行于 y 轴的直线 L'，L'与椭圆曲线再次相交于 S 点，S 则被定义为 P 与 Q 的“点加”之和，记作 $S=P\oplus Q$。由图 4.5 可见，S 与 R 关于 x 轴对称。

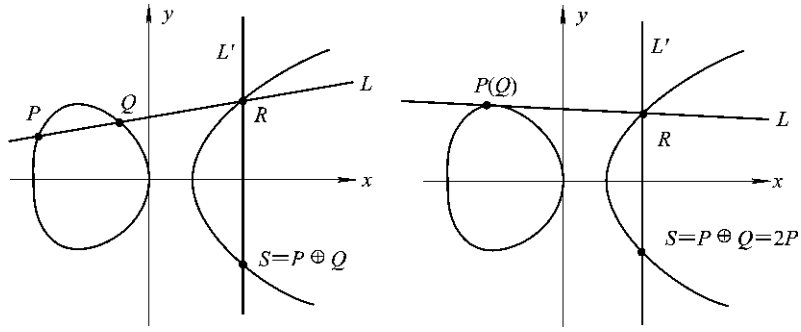

图 4.5 椭圆曲线“点加”的定义

如果把椭圆曲线上点的集合看做一个群，把上述方式定义的“点加”运算看做是群元的基本运算，为了群的封闭性，集合还包括无限远点 Φ。这是因为直线 L'可以视为无限远点 Φ 与 R 点的连线，按上述“点加”的定义，$\Phi\oplus R$ 应当是 Φ 与 R 的连线 L'与椭圆曲线交点 S 的关于 x 轴的对称点，那就是 R 点自己。可见，Φ 是群的恒元，任意点与 Φ 的点加等于自己，即 $R\oplus\Phi=R$。

因为 R 与 S 关于 x 轴对称，所以 $R\oplus S=\Phi$，表明它们互为逆元。若任意两点的点加之和等于无穷远点，则这两点互为逆元。

设 $P\oplus Q$ 点的坐标为(x_3,y_3)，用解析几何方法不难得到 $P\oplus Q$ 点的坐标：

$$\begin{cases} x_3=\lambda^2-x_1-x_2 \\ y_3=\lambda(x_1-x_3)-y_1 \end{cases} \tag{4-64}$$

式中

$$\lambda=\begin{cases} \dfrac{y_2-y_1}{x_2-x_1} & \text{当 } P\neq Q \text{ 时} \\ \dfrac{3x_1^2+a}{2y_1} & \text{当 } P=Q \text{ 时} \end{cases}$$

特别是当 $P=Q$ 时，连接 PQ 的直线 L 变成了过 P 点的切线，这时 $P\oplus Q=2P$。计算 $2P$ 点的坐标的公式不变，只是 λ 不同而已。

4.7.3 Z_p 域上的椭圆曲线

设 p 是一个大于 3 的素数，在 Z_p 域上，椭圆曲线的定义是：

$$y^2=x^3+ax+b \bmod p \tag{4-65}$$

且 a，b 满足：

$$4a^3+27b^2\neq 0 \bmod p \tag{4-66}$$

则解集$(x,y)\in Z_p\times Z_p$ 和一个无穷远点 Φ 构成 Z_p 域上的椭圆曲线。也就是说，除了无穷远点外，解集(x,y)不仅要求是椭圆曲线上的点，而且要求必须是$[0,p-1]$区间的整数

值。此外还要求按照模运算规律把满足椭圆曲线上远处的整数点移入$(p-1)\times(p-1)$区间，形成某种离散分布的形式。满足上述条件的点的集合，称为 Z_p 域上椭圆曲线的解集，用 E 表示。

同时定义椭圆曲线的判别式为

$$\Delta(E) = -16(4a^3 + 27b^2) \tag{4-67}$$

椭圆曲线的 j 不变量为

$$J(E) = \frac{-1728(4a)^3}{\Delta(E)} \tag{4-68}$$

离散域椭圆曲线点加运算的定义同上。

【例 9】 $y^2 = x^3 + x + 1$ 是 Z_{23} 上的椭圆曲线，计算$(3, 10)\oplus(6, 4)$的值。

解： 首先我们看到 23 是模 4 余 3 的素数，因此题目有效。

因为

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = -2 \bmod 23 = 21$$

$$x_3 = \lambda^2 - x_1 - x_2 = 21^2 - 3 - 6 \bmod 23 = 18$$

$$y_3 = \lambda(x_1 - x_3) - y_1 = 21(3 - 18) - 10 \bmod 23 = 20$$

所以

$$(3, 10)\oplus(6, 4) = (18, 20)$$

【例 10】 $y^2 = x^3 + x + 1$ 是 Z_{23} 上的椭圆曲线，求出它的解集。

解： 实际上这是一个二次剩余方程：

$$y^2 = x^3 + x + 1 \bmod 23$$

令 $z = x^3 + x + 1$，就有 $y^2 = z \bmod p$ 的形式。二次剩余的平方根有两个，即 $y_1 = \sqrt{z} = z^{\frac{p+1}{4}} \bmod p$ 和 $y_2 = p - y_1$。如：

① $x = 0$ 时，$z = 1$，得到 $y_1 = z^6 = 1$，$y_2 = 23 - 1 = 22$，即$(0, 1)$和$(0, 22)$；

② $x = 1$ 时，$z = 3$，得到 $y_1 = 3^6 \bmod 23 = 16$，$y_2 = 23 - 16 = 7$，即$(1, 16)$和$(1, 7)$。

同理可推出 $x \in [0, 22]$ 中的全部解，见表 4.6。

表 4.6　椭圆曲线 $y^2 = x^3 + x + 1 \bmod 23$ 的全部解

x	0	1	3	4	5	6	7	9	11	12	13	17	18	19
y_1	1	7	10	0	4	4	11	7	3	4	7	3	3	5
y_2	22	16	13	23	19	19	12	16	20	19	16	20	20	18

在模 23 运算下，$(4, 0)$与$(4, 23)$其实是同一个点，曲线上有 27 个点，连同无穷远点 Φ，共有 28 个点。它们的分布如图 4.6 所示。

关于 GF(q)域上椭圆曲线解域点的数目 N，经 Artin 提出，Hasse 证明，N 的估计值满足：

$$|N - (p + 1)| \leqslant 2\sqrt{p}$$

上例中，$p = 23$，$N \leqslant 23 + 1 + 9 = 33$，估计正确。

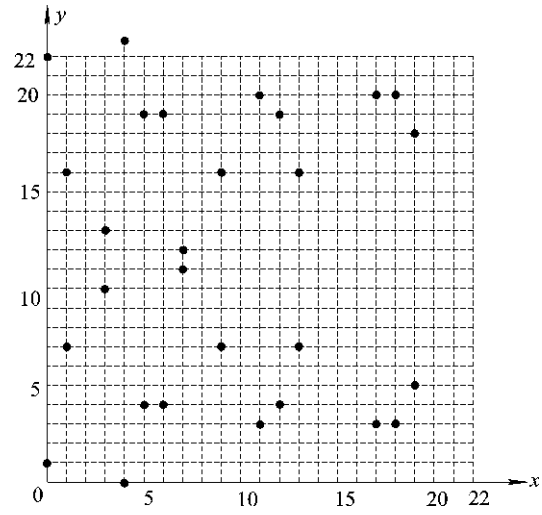

图 4.6 Z_{23}上椭圆曲线 $y^2=x^3+x+1$ 的解的分布

4.7.4 GF(2^m)域上的椭圆曲线

在有限域 GF(2^m)上，椭圆曲线的定义是：

$$y^2+xy=x^3+ax^2+b \qquad a,b,x,y\in \mathrm{GF}(2^m) \tag{4-69}$$

GF(2^m)的域元素有$\{0,1,\alpha,\alpha^2,\alpha^3,\cdots,\alpha^{q-2}\}$，$q=2^m$。其中 α 是 GF(2^m)域的本原元素，是本原多项式 $f(x)$的根，它们遵循域运算的法则。

GF(2^m)域限定了椭圆曲线的参数和取值都只能在域元素的集合之中。

GF(2^m)域中椭圆曲线元素“点加”的定义不变，但解析表达式与 GF(p)域的不同。

设

$$P(x_1,y_1)\oplus Q(x_2,y_2)=G(x_3,y_3)$$

则

$$\begin{cases} x_3=\lambda^2+\lambda+x_1+x_2+a \\ y_3=\lambda(x_1+x_3)+x_3+y_1 \end{cases} \tag{4-70}$$

式中：

$$\lambda=\begin{cases} \dfrac{y_1+y_2}{x_1+x_2} & \text{当 } P\neq Q \\ \dfrac{x_1^2+y_1}{x_1} & \text{当 } P=Q \end{cases}$$

将 λ 代入得到：

① 对于 $P\neq Q$，有：

$$\begin{cases} x_3=\left(\dfrac{y_1+y_2}{x_1+x_2}\right)^2+\dfrac{y_1+y_2}{x_1+x_2}+x_1+x_2+a \\ y_3=\left(\dfrac{y_1+y_2}{x_1+x_2}\right)(x_1+x_3)+x_3+y_1 \end{cases} \tag{4-71}$$

② 对于 $P=Q$，有：

$$\begin{cases} x_3 = \left(x_1 + \dfrac{y_1}{x_1}\right)^2 + \left(x_1 + \dfrac{y_1}{x_1}\right) + a = x_1^2 + \dfrac{b}{x_1^2} \\ y_3 = x_1^2 + \left(x_1 + \dfrac{y_1}{x_1}\right)x_3 + x_3 \end{cases} \tag{4-72}$$

【例 11】 椭圆方程为 $y^2+xy=x^3+\alpha^8x^2+\alpha^2$，$\alpha$ 是 $\mathrm{GF}(2^4)$域的本原元素，本原多项式为 $f(x)=x^4+x+1$，试求出它的解集。

解： 由于 α 是 $f(\alpha)=\alpha^4+\alpha+1=0$ 的根，因此域元素有下列关系成立：

$\alpha^0=1\rightarrow(0001)$；　　$\alpha^1=\alpha\rightarrow(0010)$；

$\alpha^2=\alpha^2\rightarrow(0100)$；　　$\alpha^3=\alpha^3\rightarrow(1000)$；

$\alpha^4=\alpha+1\rightarrow(0011)$；　　$\alpha^5=\alpha^2+\alpha\rightarrow(0110)$

$\alpha^6=\alpha^3+\alpha^2\rightarrow(1100)$；　　$\alpha^7=\alpha^3+\alpha+1\rightarrow(1011)$

$\alpha^8=\alpha^2+1\rightarrow(0101)$；　　$\alpha^9=\alpha^3+\alpha\rightarrow(1010)$

$\alpha^{10}=\alpha^2+\alpha+1\rightarrow(0111)$；　　$\alpha^{11}=\alpha^3+\alpha^2+\alpha\rightarrow(1110)$

$\alpha^{12}=\alpha^3+\alpha^2+\alpha+1\rightarrow(1111)$；　　$\alpha^{13}=\alpha^3+\alpha^2+1\rightarrow(1101)$

$\alpha^{14}=\alpha^3+1\rightarrow(1001)$；　　$\alpha^{15}=1=\alpha^0\rightarrow(0001)$

用尝试法不难找到能满足原方程的解集及点加关系，见表 4.7。

表 4.7　$\mathrm{GF}(2^4)$域上椭圆曲线 $y^2+xy=x^3+\alpha^8x^2+\alpha^2$ 的全部解

	P	$2P$	$3P$	$4P$	$5P$	$6P$	$7P$	$8P$	$9P$	$10P$	$11P$	$12P$	$13P$	$14P$	$15P$
x	α^3	α^1	α^9	α^8	α^7	α^0	α^{13}	0	α^{13}	α^0	α^7	α^8	α^9	α^1	α^3
y	α^1	α^2	α^{10}	α^0	α^{11}	α^0	α^1	α^1	α^{12}	0	α^8	α^2	α^{13}	α^5	α^9

表 4.7 中，$P(\alpha^3,\alpha)$为基点，$16P$ 是无限远点。

4.7.5　关于椭圆曲线的群和阶

1. 椭圆曲线解域构成阿贝尔群

椭圆曲线上的所有点，连同无穷远点 Φ，称做椭圆曲线的解域，记作 E。解域中点的个数 n 叫做椭圆曲线的阶，记作 $n=\sharp E$。E 上的点满足群的四个条件，首先是封闭性，按照上面所定义的$\oplus$运算，任何两点相加，在 E 内必能找到 $P\oplus Q$ 点(证明从略)。如果解域是连续取值的实数域，则不难理解其封闭性，但当解域为离散的有限域整数点集合 $Z_p\times Z_p$，可以想象，这是多么的奇妙！令人难以置信！然而这却是正确的结论。

其他三条都很直观：恒元就是无穷远点，任意点 $P\oplus\Phi=P$；由逆元 $P\oplus(-P)=\Phi$，立刻知道互逆两点是 x 坐标相同、y 坐标符号相反的两点，而椭圆曲线正是以 x 轴为对称的；结合律成立也没有问题，于是 E 上点构成群。

按椭圆曲线的“点加”定义，交换律 $P\oplus Q=Q\oplus P$ 自然成立，因此它是阿贝尔群。

2. 基点和基点的阶

P 是 E 上的一个点，定义能使 $nP=P+P+\cdots+P=\Phi$ 成立的最小正整数 n 为点 P 的

阶，而这个 P 叫做基点。显然，不同点的阶是不同的，它反映了不同点的循环性质和周期。

【例 12】 Z_5 域上 E 的方程式为 $y^2=x^3+1$，$P=(2,3)$为基点，求 n 的值。

解：从 $2P=P+P$ 作起，求 P 点切线方程。因为

$$\left.\frac{\mathrm{d}y}{\mathrm{d}x}\right|_P=\left.\frac{3x^2}{2y}\right|_{(2,3)}=2$$

所以

$$y-3=2(x-2)$$

即 $y=2x-1$。求它与 $y^2=x^3+1$ 的交点，联立解得 $x=0$，$y=-1$，因此 $2P=(0,1)$。

再求 $4P=2P+2P$，可以用公式直接写出结果。因为

$$\lambda=\frac{3x_1^2+a}{2y_1}=\frac{3\times 0^2+0}{2\times 1}=0$$

所以

$$x_3=\lambda^2-x_1-x_2=0-0-0=0$$

$$y_3=\lambda(x_1-x_3)-y_1=0-1=-1$$

最后再将 $4P=(0,-1)$和 $2P=(0,1)$两点相加。显然，$4P+2P=6P=\Phi$(这两点关于 x 轴对称)，因此 $P=(2,3)$为基点时，椭圆曲线的阶 $n=6$；而当 $2P=(0,1)$为基点时，椭圆曲线的阶 $n=3$。具体见表 4.8。

表 4.8 例 12 椭圆曲线上点的数乘关系

	$2P$	$4P$	P	$5P$	$3P$	$6P$
x	0	0	2	2	4	Φ
y	1	4	3	2	0	

【例 13】 在 Z_{23} 上的椭圆曲线为 $y^2=x^3+12x+1$，基点为 $P=(0,1)$，求循环阶。

解：因为

$$\lambda=\frac{3x_1^2+a}{2y_1}=6$$

$$x_{2P}=\lambda^2-2x_1=36 \bmod 23=13$$

$$y_{2P}=\lambda(x_1-x_{2P})-y_1=6(0-13)-1 \bmod 23=13$$

所以 $2P=(13,13)$。同样可以求得各个 nP 如表 4.9 所示。

表 4.9 Z_{23} 上的椭圆曲线 $y^2=x^3+12x+1$ 的数乘循环数据

	P	$2P$	$3P$	$4P$	$5P$	$6P$	$7P$	$8P$	$9P$	$10P$	$11P$	$12P$	$13P$	$14P$	$15P$	$16P$
x	0	13	5	3	6	19	17	18	17	19	6	3	5	13	0	Φ
y	1	13	5	15	17	2	9	0	14	21	6	8	18	10	22	

因为模 23 下 $15P=(0,22)=(0,-1)$，与 $P=(0,1)$ 以横轴为对称，所以 $16P=15P\oplus P=\Phi$(无穷远点)，表明基点 $P=(0,1)$的循环阶是 $n=16$。

4.7.6 椭圆曲线上的密码系统

1. ElGamal 公钥体系

1) 系统参数

设 E 是定义在 Z_p($p>3$ 的素数)上的椭圆曲线，令 $\alpha\in E$，由 α 生成的子群 H 满足离散

对数计算复杂要求，计算 $\beta=a\alpha$，其中 a 可任选。则私有密钥为 $k_2=a$，公有密钥为 $k_1=(\alpha,\beta,p)$。

2）加密算法

对于明文 x，随机选取正整数 $k\in Z_{p-1}$，有

$$E_{k_1}(x,k)=(y_1,y_2) \tag{4-73}$$

其中：

$$y_1=k\alpha,\ y_2=x+k\beta \tag{4-74}$$

3）解密算法

解密算法为

$$D_{k_2}(y_1,y_2)=y_2-ay_1=x+k\beta-ak\alpha=x+ka\alpha-ak\alpha=x \tag{4-75}$$

这种思路与离散对数公钥系统十分相似。离散对数公钥系统是把一个数 x 与它的 a 次方 y 作为公钥，而把 a 作为私钥，利用的是计算离散对数的复杂性。此处是把椭圆上的一个点 x 与它的 a 倍的那个点 y 作为公钥，而把 a 作为私钥，利用的是计算椭圆除法的复杂性。根据这个相似性，人们也往往把椭圆曲线问题称为椭圆曲线离散对数。

【例 14】 取椭圆曲线 E 为 $y^2=x^3+x+1 \bmod 23$，试对明文 $x=(5,4)$加密。并分析收到密文后如何解密。

解：设 $\alpha=(6,4)$，取 $a=3$，有 $\beta=3\alpha=(7,12)$。

若 A 想加密明文 $x=(5,4)$，则首先选取 $k=2$，那么：

$$y_1=k\alpha=(13,7)$$

$$y_2=x+k\beta=(5,4)\oplus 2(7,12)=(5,4)\oplus(17,3)=(5,19)$$

所以密文为

$$y=(y_1,y_2)=((13,7),(5,19))$$

B 收到密文后，解密如下：

$$x=y_2-ay_1=(5,19)\oplus[-3(13,7)]=(5,4)$$

2. Diffie-Hellman 椭圆公钥体系

Diffie-Hellman 最初以离散对数的方式实现了 A 与 B 的密钥交换问题。其实密钥交换问题也可以基于椭圆曲线实现，思路完全相同。

A 选椭圆曲线 E 上任意点 P 为公钥，再任选一个小于循环阶 n 的数值 a，作为私钥秘密保管，计算 aP，发给 B；B 选私钥 b，计算 bP，发给 A。之后两人便有了公共的会话密钥 abP：A 通过计算 $a(bP)$获得，B 通过计算 $b(aP)$获得。

3. Massey-Omura 椭圆公钥体系

已知有限域 GF(q)，用户 A 选一加密密钥 $e_A(0\leqslant e_A\leqslant N)$，满足$(e_A,q-1)$，并求出模 $q-1$ 的逆元 d_A，使 $e_Ad_A=1 \bmod (q-1)$。

同样，B 选 e_B 使$(e_B,q-1)=1$，同时计算出逆元 d_B，使 $e_Bd_B=1 \bmod (q-1)$。

若 A 想给 B 发送消息 m，应发送 m^{e_A}（相当于 A 给箱子上了一道锁）。B 收到后，返还给 A 的是$(m^{e_A})^{e_B}=c$（相当于 B 给箱子再加上一道锁）。A 收到 c 后，计算 $c^{d_A}=(m^{e_Ae_B})^{d_A}=m^{e_B}$（相当于 A 打开了自己的锁，但仍被 B 锁着）。之所以能这样做，是基于指数上的两个因子可以交换次序。B 再收到 m^{e_B}后，计算$(m^{e_B})^{d_B}=m$，从而获得明文（相当于 B 打开了

自己的锁)。这是基于 RSA 的算法。现将同样的思路在椭圆曲线上实现。

设明文 m 嵌入 E 上的点 P_m，E 上的点数目 N 很大，选择 e 使$(e, N)=1$，并求出逆元 d，使 $ed=1 \bmod N$。A 发送 m 给 B，首先送去 $e_A P_m$；B 收到后计算 $e_B(e_A P_m)$返回；A 计算 $d_A(e_B e_A P_m)=e_B P_m$，再发给 B，于是 B 得到 $d_B(e_B P_m)=P_m$。

之所以能这样做，是基于椭圆曲线的解集构成可交换群。

4.7.7 椭圆曲线密码体制的安全性

1. 椭圆曲线密码体制的特点

椭圆曲线密码体制有如下特点：

(1) 椭圆曲线是不同于大整数分解和离散对数的另一类算法。

(2) 椭圆曲线资源丰富，为安全性提供了额外的保证。

(3) 复杂性高，p 为 160 bit 的椭圆曲线的复杂性就相当于 n 为 1024 bit 的 RSA，相对于 RSA 而言，所计算的数值较小，适用于智能卡。

(4) 签名和解密速度比 RSA 快，签名的验证和加密比 RSA 慢。

(5) 至今对椭圆曲线提出的攻击(分析)较少，表明安全性高。

2. 安全的椭圆曲线

椭圆曲线公钥密码体制的安全性是建立在椭圆曲线离散对数的基础之上的。所谓椭圆曲线离散对数，指的是计算点乘 $P=nG$ 是容易的，但当 n 很大时，由 P 和 G 求 n 的逆运算是极其复杂的。而 n 不可能大于基点 G 的阶 $\sharp E$，所以为了安全，要求椭圆曲线解域上基点的阶值 $\sharp E$ 尽量大。据相关文献论述[14]，椭圆曲线的阶应含有至少 40 位十进数的大素因子，才能防止现有的各种攻击。为此，作为模的大素数往往取到 160 bit 以上，才有可能存在如此大的阶值。实用的椭圆曲线，常取 192 bit 的大素数为模来设计椭圆密码体制。

然而，椭圆曲线的模足够大只是椭圆曲线的阶值大的必要条件，二者之间并无简单的对应关系，在如此大的 Z_p 域中寻找基点并计算阶值，无疑是不容易的事。但是为了系统的安全，设计者却不得不考察椭圆曲线的阶值。如果椭圆曲线的阶值 $\sharp E$ 是一个大素数，则椭圆曲线解域中除了无穷远点的任意点都可以作为基点，其循环阶均为 $\sharp E$；或者虽然椭圆曲线的阶值 $\sharp E$ 是一个大的合数，但它含有大的素数因子，则在这个素因子为循环阶的子群中，也会有较大的循环阶，这样的椭圆曲线就比较安全。而如果椭圆曲线的阶值 $\sharp E$ 不大，或者虽然它是一个大的合数，但它可分解为许多小的素数因子，那么这样的椭圆曲线就不安全。一句话，我们希望椭圆曲线的阶是大素数或者含有大素因子。

其次，并不是所有大阶值的椭圆曲线都可以应用到公钥密码体制中，对于一些特殊类型的椭圆曲线，存在已知的攻击方法，应避免选用。它们是[15]：

(1) 基点 G 的阶 $\sharp E$ 等于有限域大小 p 的曲线是异常椭圆曲线，存在 SSSA 攻击算法。

(2) 基点 G 的阶 $\sharp E$ 是 $p-1$ 的因子，这类曲线有 FR 攻击算法。

(3) 基点 G 的阶 $\sharp E$ 虽不是 $p-1$ 的因子，却是 p^k-1 的因子，且 k 较小，对于这类曲线，存在 MOV 攻击算法和 FR 攻击算法，如果要使用，应当选用 $k>20$。

(4) 对于域 $\mathrm{GF}(2^m)$上的椭圆曲线，存在 Weil 下降攻击算法，为此，应选 m 为素数。

3. 椭圆曲线密码体制中参数的选取

对于素数为模的椭圆曲线，寻找安全椭圆曲线，就是确定一组适当的椭圆曲线参数(a, b, p)，使阶值满足安全要求。一般有两种方案。

1) *CM* 算法

根据设定的素数域(p 值)首先选定一个安全的阶，然后构造具有该阶值的椭圆曲线。此方法最早由 Aktin 和 Morain 提出，立刻引起了广泛关注，IEEE P1363 也采用了该算法。它被称为复乘法(CM 算法)，有简单易行的特点。但也有人认为它产生的 E 与虚二次域的某个阶有内在联系，存在安全隐患[16]。

CM 算法的基本步骤如下(细节请查阅相关文献)：

(1) 给定素数 p，随机选取 t 和 s，求解满足 $4p=t^2+Ds^2$ 的最小的基本判别式 D，并判断 D 是否满足 $D=3 \bmod 4$。

(2) 检查 $p+1\pm t$ 的素性，若不为素数，回到(1)。

(3) 构造类多项式 $H_D(x)$。

(4) 求解方程 $H_D(x)=0 \bmod p$ 的根 j_0，j_0 为 j 不变量。

(5) 计算 $k=\dfrac{j_0}{1728-j_0}$，则椭圆曲线为 $E_1(y^2=x^3+3kx+2k)$。

(6) 检查椭圆曲线 E_1 的阶，若不为 $p+1-t$，则随机选取 c(c 是 F_p 中的非平方数)，构造与 E_1 互为扭曲的椭圆曲线 $E_2(y^2=x^3+c^2ax+c^3b$，其中 $a=3k$，$b=2k)$。

2) *SEA* 算法

首先随机产生一条椭圆曲线(随机生成椭圆曲线参数 a 和 b)，然后计算它的阶，判断是否安全。若不安全，则放弃，重新生成一条再来测试，直至满意为止。此法的关键在于如何计算阶，Schoof[17][18]首先提出了利用 l-torsion 子群中的 Frobenius 映射和中国剩余定理来求阶的方法，后被 Elkies 和 Atkin 加以改进，形成所谓的 SEA 方法[19]。此法虽然麻烦，但安全性好，是目前普遍采用的方法。

SEA 算法步骤大体如下(细节请参考相关文献)：

(1) 随机选取曲线参数 a，$b\in F_p$，$a\cdot b\neq 0$，$\Delta=4a^3+27b^2\neq 0$；

(2) 对 Atkin 素数，计算 $t \bmod l$ 的信息；对 Elkies 素数，判断除子多项式 $h(x)$外的参数在域中是否有根。若有根 x_p 且勒让德符号 $\left(\dfrac{x_p^3+ax_p+b}{p}\right)=1$，则回到(1)；否则，计算 $t \bmod l$，若 $p+1=t \bmod l$，则回到(1)。

(3) 计算 $\sharp E(F_p)$，并进行素性测试，若 $\sharp E(F_p)$为合数，则回到(1)；否则，输出椭圆曲线方程 $y^2=x^3+ax+b$。

对于 $GF(2^m)$域上的椭圆曲线，一般采用降阶法来产生安全的椭圆曲线：如果 m 能被一个比较小的整数 d 整除，我们首先在有限域 $GF(2^d)$上寻找椭圆曲线 E'，并计算其阶，根据此值，利用 Weil 定理，计算该曲线在其扩域 $GF(2^m)$上的阶，若此阶符合安全标准，再找曲线 E'在 $GF(2^m)$ 域上的嵌入 E，则 E 为所需的安全曲线。

4. 安全椭圆曲线的参数举例

(1) 阶为大素数的椭圆曲线参数：

$p=1452046121366725933991673688168680114377396846591$

$a=-3$

$b=49$

$\#E=145204612136672593399167314625889010471353372030 3$

$G=(0,7)$

(2) **阶含有大素因子的椭圆曲线参数**：

$p=2^{160}-47=1461501637330902918203684832716283019655932542929$

$a=150109981375978573244826447265299164247145 85$

$b=213910743184753764084459666168311750783430782$

$\#E=1461501637330902918203684313125075552876176419731$

$=11\times 893498881393192080 91\times 1487005613311253628184160731$

(3) GF(2^m)**域的椭圆曲线参数**：

$y^2+xy=x^3+ax^2+b \quad a,b,x,y\in \mathrm{GF}(2^m)$

$m=167,a=2,b=141$

$\#E=4\times(4676805239\ 4588893382\ 5179172635\ 2112434071\ 5321943773)$

习 题 4

1. **利用下列数据实现** RSA **算法的加密和解密**：

(1) $p=3$；$q=11$，$e=7$；$m=5$。

(2) $p=5$；$q=11$，$e=3$；$m=9$。

(3) $p=7$；$q=11$，$e=17$；$m=8$。

(4) $p=11$，$q=13$，$e=11$；$m=7$。

2. **假设字符编码机制为**：A=1，B=2 …… $Z=26$。**若明文字符为** F，RSA **中，取** $e=5$，$d=77$，$n=119$，**求加密后的密文。并验证解密后能得到的明文** F。

3. **使用** RSA **的系统，已知密文** $C=10$，**该用户公钥** $e=5$，$n=35$，**明文** M **等于多少？**

4. RSA **体系中已知** $n=2548903037$，$\Phi(n)=2548792896$，**试分解** $n=pq$。

5. RSA **体系中已知** $e=17$，$d=89$，$n=1591$，**求** p **和** q。

6. B **用下述方法加密消息** M，**发送给** A：

(1) A **选择两个大素数** P **和** Q。

(2) A **公布其公钥** $N=PQ$。

(3) A **计算** P' **和** Q'，**使得** $PP'\equiv 1(\mathrm{mod}\ Q-1)$，**且** $QQ'\equiv 1(\mathrm{mod}\ P-1)$。

(4) B **计算** $C=M^N(\mathrm{mod}\ N)$，**发给** A。

(5) A **求解** $M=C^{P'}(\mathrm{mod}\ Q)$**或** $M=C^{Q'}(\mathrm{mod}\ P)$ **均可得到** M。

试回答：

(1) **这种方法的原理**。

(2) **它与** RSA **有何不同？**

(3) **与此相比，**RSA **有何优点？**

7. **根据中国剩余定理提出了一个素数测试方法**：n **是素数当且仅当** n **能整除**(2^n-2)。**然而中国古代数学家却没有测试** $n=341$ **这个例外，**341 **不是素数，但是却能整除** $2^{341}-2$。

试证明 $2^{341}\equiv 2 \bmod 341$(即必要条件不成立)。

(提示：可用同余定理证明，不必计算 2^{341}。)

8. 使用 ElGamal 算法，进行以下操作：

(1) 公用素数 $p=97$，其本原根 $g=5$。若 B 的公钥 $\beta=44$，A 选择的随机数 $k=36$，确定 $m=3$ 的密文。

(2) 公用素数 $p=71$，其本原根 $g=7$，公钥 $\beta=3$。若 A 选择的随机数 k 使得 $m=30$ 的密文是 $C=(59, y_2)$，则整数 y_2 是多少？

9. 基于 Pohlig-Hellman 算法，如果 $p=8101$，$x=6$，$y=7531$，满足 $y\equiv x^r \pmod p$，求 r 的值。

10. 椭圆曲线上同在一条直线上的三个点的和是什么？

11. 取 E 为 Z_{11} 上的椭圆曲线 $y^2=x^3+x+6$，基于 ElGamal 公钥体系，设 $\alpha=(2, 7)$，Bob 的私钥是 7，若 Alice 选择随机数 $k=3$，那么要加密明文 $x=(10, 9)$，应如何运算？Bob 收到密文如何解密？

12. 对于有理数域上的椭圆曲线 E：$y^2=x^3-36x$，令 $P_1=(-3, 9)$，$P_2=(12, 36)$，P_1，$P_2\in E$，计算 P_1+P_2 和 $2P_1$。

13. 背包体制，设超递增序列 $B=\{24, 64, 128, 255, 500, 999\}$，选 $p=2000$，$w=69$。对明文 $M=101010$ 加密的密文是什么？并给出解密过程。

实践练习 4

实验题目： RSA 公开密钥体系的构建与加密、解密练习。

实验平台： Mathematica 4.0。

实验内容： 生成大素数，构件 RSA 密码系统，并对明文进行加密与解密。

实验步骤：

(1) 大素数的生成与检测。

(2) 由大素数 p 和 q 生成 RSA 公钥 n、e 与私钥 d。

(3) 将已知的明文变换成数字形式(比如按字母表序号)。

(4) 对数字明文进行加密运算。

(5) 对密文进行解密运算，并变换回字符形式。

第5章 签名与认证

信息技术带来现代社会变革的一个重大方面是电子商务，它极大地促进了传统商务模式的改变和结构的更新。而电子商务的发展，对信息安全技术又提出了多方位的新要求，主要表现在形形色色的签名与认证需求方面。密码技术近年来的巨大发展，也正集中体现在这些方面。

5.1 数字签名

A 收到 B 以明文方式送来的信息后，A 如何鉴别是 B 所发而非别人伪造或篡改的呢？又如公钥密码系统中，各用户从网络上获取对方的公开密钥，如何保证它没有被篡改或替代呢？倘若 C 用自己的公钥替换了 B 的公钥，那么它就可以用自己的私钥解读所有人发给 B 的密文。再如，单钥体系中，A、B 两方均掌握密钥 K，如果没有签名，一旦发生争议就说不清楚，因为 B 可以修改 A 发来的密文，A 也可以自己修改原文以赖账，法律上缺乏判定谁是谁非的凭证。

传统的(手写)签名借助纸张将文件的内容与签发人的笔迹(认可凭证)有效地结合在一起，使文件具有了可认证性。然而，由于电子文档的易拷贝性和可粘贴性，使机械地照搬手写签名到电子文档上的做法失效，必须引入功能相似，但方式不同的有效认证方式。

数字签名是含有发送方身份的一段数字或代码串。它应有以下特征：

(1) 签名是可以验证的，收到签名的人容易由签名确定来信人。

(2) 签名是不可伪造的，除了签名者之外的任何人无法实现这个签名。

(3) 签名是不可重用的，一个签名只对一个文件生效，无法用于其他文件。

(4) 签名是不可改变的，一旦签名发出便不能再作修改。

(5) 签名是不可抵赖的，存在某种方法充分证明该签名确为发信人所为。

用发信人私钥加密所发信件得到的密文，就是具有上述特征的一种数字签名。

为了适应多种不同的用途，为了使用起来更方便更安全，滋生出了一系列不同形式的数字签名，下文将陆续介绍。它们在电子商务和信息网络中发挥着不可替代的作用。

数字签名的主要功能有：

(1) 识别来信(函)的正确信源。

(2) 验证发信人的合法身份。

(3) 检查信函内容的完整性。

(4) 获得行为不可否认的证据。

(5) 标示知识产权的印记。

5.1.1 RSA 签名体系

RSA 签名体系与 RSA 加密算法相同，但用法不同。

系统参数

取两个大素数 p 和 q，计算 $n=p\cdot q$ 和 $\phi(n)=(p-1)(q-1)$，随机选择整数 d 使 $\gcd(d,\phi(n))=1$，求出模 $\phi(n)$ 的逆元 e，使它满足 $ed=1 \bmod \phi(n)$。取公开密钥为 $k_1=(n,e)$；私有密钥为 $k_2=(p,q,d)$。

签名算法

用私钥"加密"明文 x，得到的结果 y 叫做 x 的签名，即

$$\mathrm{Sig}_{k_2}(x)=x^d \bmod n=y \tag{5-1}$$

验证算法

$$\mathrm{Ver}_{k_1}(y)=y^e \bmod n\equiv x \tag{5-2}$$

首先 y 只能是 x 的签名(是 x 的加密结果)，它不能用于其他文档；其次，y 是用发信人的私钥"加密"所得，别人做不出这样的签名。任何人都可以用发信人的公钥解出 x，与原来的 x 对照，即可证明以上两点。

通常不仅要认证签名，还要加密明文，不让合法收信人以外的其他人看到明文 x，这就要进行双重"加密"。如 A 给 B 发信，有两种方案：

(1) 先加密再签名：A 做 $z=E_B(x)$，$y=\mathrm{Sig}_A(z)$的运算，把(z,y)传输给 B；B 收到(z,y)后，用自己的私钥解读密文，即做 $x=D_B(z)$的运算，再用 A 的公钥验证签名 $\mathrm{Ver}_A(z,y)$，看是否满足 $y^{e_A}=z$。

这样做的不安全之处在于如果窃听者 C 收到(z,y)，它可以用自己的密文 $z'=E_B(x')$ 替代 z，并对 z'作自己的签名：$\mathrm{Sig}_C(z')=y'$，再用 y'代替 y。而 B 收到(z',y')后，可能会做出发信人是 C 的误解，因为用 C 的公钥(e_C,n')能够证明签名 $\mathrm{Ver}_C(z',y')$，即 $y'^{e_C}=z' \bmod n'$，从而相信了 C 发来的密文 x'，结果受了骗。

(2) 先签名再加密：A 先做 $\mathrm{Sig}_A(x)=y$ 的运算，再做连明文带签名一起加密 $E_B(x,y)=z$ 的运算，发送 z 给 B；B 收到 z 后先解密得到签名，再进行验证。

这种方案中，窃听者 C 即使收到 z，也因无 A 的私钥而不能解译，所以比较安全。

5.1.2 ElGamal 签名方案

1985 年，ElGamal 基于离散对数提出了一种签名方案。

系统参数

p 是大素数，g 是 Z_p^* 的一个生成元(即本原根)。定义密钥为

$$K=\{(p,g,y,x):\ y=g^x \bmod p\},\ x\in Z_p^*$$

其中，公开密钥 k_1 为 y、p、g；私有密钥 k_2 为 x。

签名算法

签名者拥有$(k_1k_2)=(p,g,y,x)$，随机数 $k\in Z_p^*$ 和待签消息 m。生成的签名为

$$\mathrm{Sig}_{k_2}(m,k)=(r,s) \tag{5-3}$$

这里，对 r 和 s 有

$$r = g^k \bmod p, \quad s = (m - xr)k^{-1} \bmod (p-1) \tag{5-4}$$

随机数 k 用以生成签名中的 r 部分，而明文用以生成签名中的 s 部分。

验证算法

验证者有公钥 $k_1=(p, g, y)$，收到的明文 m 和签名 (r, s)，从而验证算法为

$$\mathrm{Ver}_{k_1}(r, s) = y^r r^s \equiv g^m \bmod p \tag{5-5}$$

实际上，因为

$$ks = (m - xr) \bmod (p-1)$$

即

$$ks=\lambda(p-1)+(m-xr)$$

于是有

$$g^{ks} = g^{\lambda(p-1)+(m-xr)} = g^{\lambda(p-1)} g^{(m-xr)}$$

利用欧拉定理 $g^{p-1} \bmod p=1$，就有

$$g^{\lambda(p-1)} = (g^{p-1})^{\lambda} \bmod p = 1 \bmod p$$

所以

$$g^{ks}=g^{(m-xr)} \bmod p$$

因此有

$$y^r r^s = g^{xr} g^{ks} = g^{xr} g^{(m-xr)} \bmod p = g^m \bmod p$$

使用 ElGamal 方案应注意三方面的情况：

(1) 随机数 k 不能泄露，否则用 $x=(m-sk)r^{-1} \bmod p$ 就可以在已知明文的情况下窃取私钥。

(2) k 应当每次都不同，否则，若

$$m_1 = xr + ks_1 \bmod (p-1), \ m_2 = xr + ks_2 \bmod (p-1)$$

两式相减得

$$m_1 - m_2 = k(s_1 - s_2) \bmod (p-1)$$

设 $d=\gcd\{s_1-s_2, p-1\}$，因为 $d|(p-1)$ 且 $d|(s_1-s_2)$，于是 $d|(m_1-m_2)$。再定义

$$m' = \frac{m_1 - m_2}{d}, \quad s' = \frac{s_1 - s_2}{d}, \quad p' = \frac{p-1}{d}$$

则有 $m'=ks' \bmod p'$，且 $\gcd(s', p')=1$，那么：

$$k = m's'^{-1} \bmod p', \ k = tp' + m's'^{-1} \qquad (0 \leqslant t \leqslant d-1)$$

从而通过验证 $r=g^k \bmod p$ 就可以确定 k。

(3) 由于验证只是核实等式 $y^r r^s=g^m \bmod p$ 是否成立，因此可考虑通过伪造能使上式成立的 (r, s) 来攻击。然而它的作用只不过对给定明文 m 又作了一个能通过认可的签名，既没有破译系统，又不能为其他明文生成签名，仅此而已。所以，它并不能对 ElGamal 构成威胁。

为了把明文与签名结合起来，可以有多种方式，前面所讲的方式是：

$$ks = (m - xr) \bmod (p-1)$$

其验证方程是

$$y^r r^s = g^m \bmod p$$

其他方式及其验证方程列举如下：

$mx=(rk+s) \bmod (p-1)$	$y^m=r^r g^s \bmod p$
$mx=(sk+r) \bmod (p-1)$	$y^m=r^s g^r \bmod p$
$rx=(mk+s) \bmod (p-1)$	$y^r=r^m g^s \bmod p$
$rx=(sk+m) \bmod (p-1)$	$y^r=r^s g^m \bmod p$
$sx=(rk+m) \bmod (p-1)$	$y^s=r^r g^m \bmod p$
$sx=(mk+r) \bmod (p-1)$	$y^s=r^m g^r \bmod p$
$sx=(k+mr) \bmod (p-1)$	$y^s=rg^{mr} \bmod p$
$mrx=(k+s) \bmod (p-1)$	$y^{mr}=rg^s \bmod p$
$x=(mrk+s) \bmod (p-1)$	$y=r^{mr}g^s \bmod p$
$x=(sk+rm) \bmod (p-1)$	$y=r^s g^{mr} \bmod p$
$rmx=(sk+1) \bmod (p-1)$	$y^{mr}=r^s g \bmod p$
$sx=(mrk+1) \bmod (p-1)$	$y^s=r^{mr}g \bmod p$
$(r+m)x=(k+s) \bmod (p-1)$	$y^{(m+r)}=rg^s \bmod p$
$x=[(m+r)k+s] \bmod (p-1)$	$y=r^{(m+r)}g^s \bmod p$
$sx=[k+(m+r)] \bmod (p-1)$	$y^s=rg^{(m+r)} \bmod p$
$x=[sk+(r+m)] \bmod (p-1)$	$y=r^s g^{(m+r)} \bmod p$
$(r+m)x=(sk+1) \bmod (p-1)$	$y^{(m+r)}=r^s g \bmod p$
$sx=[(r+m)k+1] \bmod (p-1)$	$y^s=r^{(m+r)}g \bmod p$

通式是：

$$Ax=Bk+C \bmod (p-1) \qquad y^A=r^B g^C \bmod p \tag{5-6}$$

5.1.3 DSS

数字签名标准(DSS)**是美国国家标准和技术研究所**(NIST)**于** 1991 **年** 8 **月公布的标准。它所采用的算法叫** DSA**，实际上是** ElGamal **的变形，签名中用的是明文的信息摘要。**

系统参数

p **是一个** 512～1024 **位的大素数，它满足离散对数难解问题；**q **是** 160 **位的素数，且** $q|p-1$；$g\in Z_p^*$ **是** Z_p **域中** q **次单位元根。定义** $K=\{(p, q, g, y, x\}: y=g^x \bmod p\}$，$h(\cdot)$**为公开的** Hash **函数。取公开密钥** $k_1=(p, q, g, y)$；**私有密钥** $k_2=(x)$。

签名算法

签名者拥有私钥 x，**对于随机数** $k\in Z_p^*$ **和待签消息** $m\in Z_p^*$，**生成的签名为**

$$\mathrm{Sig}_{k_2}(m, k)=(r, s) \tag{5-7}$$

这里：

$$r=(g^k \bmod p) \bmod q \quad (\textbf{必然小于 160 位}) \tag{5-8}$$

$$s=\{h(m)+xr\}k^{-1} \bmod q \quad (\textbf{必然小于 160 位}) \tag{5-9}$$

验证算法

验证者有公钥 $k_1=(p, g, y)$，**收到的明文** m **和签名**(r, s)，**验证算法为**

$$\mathrm{Ver}_{k_1}(m, r, s)=(g^{e_1}y^{e_2} \bmod p) \bmod q \equiv r \tag{5-10}$$

其中：

$$e_1=h(m)s^{-1} \bmod q, \ e_2=rs^{-1} \bmod q \tag{5-11}$$

实际上，由 $k=\{h(m)+xr\}s^{-1} \bmod q$ **知：**

$$g^{e_1}y^{e_2}=(g^{h(m)s^{-1}}y^{rs^{-1}} \bmod p) \bmod q=(g^{h(m)s^{-1}}g^{xrs^{-1}} \bmod p) \bmod q$$
$$=(g^{[h(m)+xr]s^{-1}} \bmod p) \bmod q=(g^{k} \bmod p) \bmod q=r$$

DSS的公布引起了学术界和商业界的激烈反应：赞成的人认为它长度小、速度快、成本低，对金融业特别有用；反对的人则认为它不与国际标准(以RSA为标准的ISO、CCITT、SWIFT等)兼容。从技术上讲，s不能等于零，要加以排除，否则危及安全性。

5.1.4 不可否认签名方案

不可否认签名方案与一般的签名方案相比较，最根本的特点是如果没有签名者的合作，签名就无法得到验证。这就防止了未经签名者同意就随意复制他的签名文件进行电子分发的可能性。有了这样的前提，一旦验证通过，签名者也就没有理由否认。

一个不可否认签名由签名算法、验证算法和否认协议三部分组成。

系统参数

设$p=2q+1$是一个大素数，这里q是素数且符合离散对数复杂性要求；$\alpha\in Z_p^*$是Z_p^*域中q次单位元根，$1\leqslant\alpha\leqslant q-1$。设$G$表示阶为$q$的$Z_p^*$的乘法子群，且定义：

$$K=\{(p,\alpha,\beta,a):\beta=\alpha^a \bmod p\}$$

其中，私有密钥为a；公开密钥为p、α、β。

签名算法

B掌握私钥a，欲对x签名，$x\in G$，则可计算：

$$y=\mathrm{Sig}_a(x)=x^a \bmod p \qquad y\in G \tag{5-12}$$

y就是B对x的签名。

验证协议

(1) A欲验证签名，可任意选取e_1，$e_2\in Z_p^*$。计算$c=y^{e_1}\beta^{e_2} \bmod p$，把它传给B。

(2) B计算$d=c^{a^{-1} \bmod q} \bmod p$，并传回A。

(3) A接收d并验证$d=x^{e_1}\alpha^{e_2} \bmod p$。 (5-13)

如果成立，便有充分必要的理由认为y是B对x的一条有效的签名。因为B用自己的私钥参与了验证过程，使A任意地选取e_1和e_2都能自恰。证明如下：

因为

$$c=y^{e_1}\beta^{e_2}=x^{ae_1}\alpha^{ae_2}$$

所以

$$d=c^{a^{-1}}=x^{e_1}\alpha^{e_2}$$

不可否认签名方案的安全性分析

首先，A是验证了$d=x^{e_1}\alpha^{e_2} \bmod p$后，才同意接收$y$作为$x$的签名的，理论上可以推知$y\neq x^a \bmod p$情况下$d=x^{e_1}\alpha^{e_2} \bmod p$的概率只有$\frac{1}{q}$，因此一个伪造的签名能使A相信的概率只有$\frac{1}{q}$。

其次，若$y=x^a \bmod p$且A遵守否认协议，但存在d和d'使：

$$d\neq x^{e_1}\alpha^{e_2} \bmod p,\quad d'\neq x^{f_1}\alpha^{f_2} \bmod p$$

则理论上可以推知$(d\alpha^{-e_2})^{f_1}=(d'\alpha^{-f_2})^{e_1} \bmod p$成立的概率为$\frac{1}{q}$。

它表明，如果B想用其他d值来否认y是他的签名，其结果就很难使等式成立(立刻会被发现)。因此，B能愚弄A的概率只有$\frac{1}{q}$。只要q充分大，B就没有理由否认自己的签名。

5.1.5 群签名

1991年，Chaum和Van Heyst基于以下问题提出群签名(Group Signature)方案：

一个公司所属各部门的计算机联网工作，各部门的打印机也联在网上。打印时，应先确定是本公司的人才可以使用，然而又要求不暴露用户的姓名。另一方面，如果打印机使用太频繁时，主管者应能够查出是谁打印了这么多东西。

一般来说，群签名系统由组、组成员(签名者)、签名接受者(签名验证者)和权威(Authority)组成，它具有以下特点：

(1) 用户组中每位合法用户都能独立对消息产生签名。

(2) 签名的接受者能验证签名的有效性。

(3) 签名的接受者不能辨认是谁的签名。

(4) 一旦发生争执，权威或组中所有成员的联合可以辨认签名者。

这里介绍一种$K—P—W$可变群签名方案。

系统参数

选择$n=pq=(2fp'+1)(2fq'+1)$，这里p、q、p'、q'为相异的大素数；g的阶为f；γ与d为整数且满足$\gamma d=1 \bmod \Phi(n)$，$\gcd\{\gamma, \Phi(n)\}=1$；$h$为安全的Hash函数；$ID_G$为用户组身份，由权威掌握。

签名组的公钥为$(n, \gamma, g, f, h, ID_G)$；签名组的私钥为$(d, p', q')$。

设ID_A为组员A的身份消息，A随机选取$s_A \in (0, f)$，并将消息$(ID_A, g^{s_A} \bmod n)$发送给权威。权威计算$x_A=(ID_G g^{s_A})^{-d} \bmod n$，并将$x_A$秘密地传送给成员A，使A掌握私有密钥$(x_A, s_A)$。同样的交换过程在权威与各个成员之间都要进行。

签名算法

对于待签消息m，组中任意成员都有权进行签名。比如A要签名，他可以随机选取整数$(r_1, r_2) \in [0, f)$，计算

$$V = g^{r_1} r_2^{\gamma} \bmod n, \quad e = h(V, m) \tag{5-14}$$

则签名为

$$(e, z_1, z_2) \tag{5-15}$$

其中：

$$z_1=r_1+s_A e \bmod f, \ z_2=r_2 x_A^e \bmod n \tag{5-16}$$

在签名过程中，虽然签名者用的是自己随机选择的r_1、r_2和s_A，但是他还必须使用来自权威的$x_A=(ID_G \cdot g^{s_A})^{-d}$，由于$g^{s_A}$掌握在权威手里，这就留下了能被权威辨认的把柄。

签名验证算法

计算

$$\overline{V}=(ID_G)^e g^{z_1} z_2^{\gamma} \bmod n \tag{5-17}$$

验证 $e=h(\overline{V}, m)$是否成立。显然在这个验证算法中，仅使用了用户组身份 ID_G 和公钥 g 和 γ，并没有涉及 A 的身份问题，所以不会暴露是谁签的名。

验证签名时，只需验证 e 是否等于 $h(\overline{V}, m)$即可。但为了说明该验证方法是合理的，则应当在此证明 $\overline{V}=V$。实际上：

$$\overline{V}=(ID_G)^e g^{z_1} z_2^{\gamma} \bmod n = (ID_G)^e g^{(r_1+s_A e)} (r_2 x_A^e)^{\gamma} \bmod n$$

$$= (ID_G)^e g^{r_1} g^{s_A e} r_2^{\gamma} x_A^{e\gamma}$$

因为 $$x_A^{e\gamma}=(ID_G g^{s_A})^{-d\gamma e}=(ID_G)^{-e} \cdot g^{-s_A e}$$

所以 $$\overline{V}=g^{r_1} r_2^{\gamma}=V$$

身份验证算法

在发生争议需验证是不是 A 签的名时，权威部门可用 A 发给他的 g^{s_A} 来验证：

$$g^{z_1} = (V \cdot r_2^{-\gamma})(g^{s_A})^e \bmod n \tag{5-18}$$

计算中所需要的 r_2 可以由 $r_2=z_2 x_A^{-e}$ 得到(因为 $z_2=r_2 x_A^e \bmod n$)，并不需要 A 提供任何信息(正因为 A 不愿暴露身份才需要权威部门出来验证)，仅由签名(e, z_1, z_2)和用户组权威部门掌握的 g^{s_A} 和 x_A，就可以验证 A 的身份。

实际上，不难证明：

$$(V \cdot r_2^{-\gamma})(g^{s_A})^e \bmod n = (g^{r_1} r_2^{\gamma} \cdot r_2^{-\gamma})(g^{s_A e}) = g^{r_1+s_A e} = g^{z_1}$$

5.1.6 多重数字签名

多重数字签名(Digital Multisignature)是一种要求多人对同一消息进行签名的数字签名方案，只有所有成员都正确完成了签名，签名才有效。根据签名过程的不同，可分为广播式(Broadcasting)和顺序式(Sequential)两种。

1. ElGamal 顺序式多重数字签名方案

顺序式多重数字签名方案的流程如图 5.1 所示。

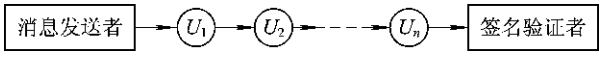

图 5.1 顺序式多重数字签名方案流程

图 5.1 中，$U_1 \cdots U_n$ 是 n 个签名者，顺序进行签名，并且每人签名时要验证上一签名者的签名的有效性，否则终止签名过程。

(1) 系统初始化。选择大素数 p，它满足 Z_p 中离散对数复杂性。g 是 GF(p)的本原元素。函数 h 是 GF(p)→GF(p)的单向 Hash 函数。每一签名者 $U_i(i=1, 2, \cdots, n)$ 的私钥是 $x_i \in [1, p-1]$，公钥是 $y_i=g^{x_i} \bmod p$，其中，p、g、y_i 和 $h(\cdot)$公开，x_i 由 U_i 密管。

(2) 签名算法。签名者 U_i 收到前一人的签名消息后$(m_1, (s_{i-1}, r_{i-1}))$后(第一个签名者取 $s_0=0$)，先验证，然后随机选取 $k_i \in [1, p-1]$，计算：

$$r_i = g^{k_i} \bmod p \quad \text{和} \quad s_i = s_{i-1} + m' x_i - r_i k_i \bmod \varphi(p) \tag{5-19}$$

这里，$m'=h(m)$。之后，将$(m_1, (s_i, r_i))$发送给下一签名者 U_{i+1}，并将 r_i 发给 U_i 以后的所有人。

(3) 验证算法。

① U_i 对 U_1，U_2，…，U_{i-1}签名有效性的验证是看等式：

$$g^{s_{i-1}}\prod_{j=1}^{i-1}r_j^{r_j}=\prod_{j=1}^{i-1}y_j^{m'} \bmod p \tag{5-20}$$

是否成立。实际上，由

$$s_i=s_{i-1}+m'x_i-r_ik_i \bmod \varphi(p)$$

就有

$$s_i=s_0+\sum_{j=1}^{i}(m'x_j-r_jk_j) \bmod \varphi(p)$$

注意，若 $s_0=0$ 就有：

$$\sum_{j=1}^{i}m'x_j=\sum_{j=1}^{i}r_jk_j+s_i \bmod \varphi(p)$$

因此，对任意 i 有下列等式恒成立：

$$g_j^{\sum_{j=1}^{i}r_jk_j+s_i \bmod \varphi(p)}=g_j^{\sum_{j=1}^{i}m'x_j \bmod \varphi(p)} \bmod p$$

即

$$g^{s_{i-1}}\prod_{j=1}^{i-1}r_j^{r_i}=\prod_{j=1}^{i-1}y_j^{m'} \bmod p$$

② 验证者 U_r 对最后的结果进行验证时，只需在上式中取 $i-1=n$ 即可：

$$g^{s_n}\prod_{j=1}^{n}r_j^{r_i}=\prod_{j=1}^{n}y_j^{m'} \bmod p \tag{5-21}$$

2. Harn 广播多重数字签名方案

Harn 广播多重数字签名方案如图 5.2 所示。

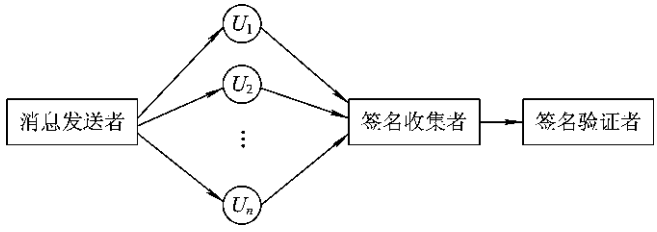

图 5.2 广播多重数字签名方案流程

（1）系统初始化。选择大素数 p，它满足 Z_p 中离散对数复杂性。g 是 GF(p)的本原元素。函数 h 是 GF(p)→GF(p)的单向 Hash 函数。每一签名者 $U_i(i=1,2,\cdots,n)$ 的私钥是 $x_i\in[1,p-1]$，公钥是 $y_i=g^{x_i} \bmod p$，其中 p、g、y_i 和 $h()$公开。

（2）单用户签名的产生。用户 U_i收到消息后，计算

$$m'=h(m) \tag{5-22}$$

然后随机选取 $k_i\in[1,p-1]$，计算

$$r_i=g^{k_i} \bmod p \tag{5-23}$$

将 r_i 发给其他各个签名者 $U_j(j\neq i)$。等收到各个用户发来的 r_j后，计算：

$$R=\prod_{i=1}^{n}r_i^{r_i} \bmod p \tag{5-24}$$

和

$$s_i=(R+m')x_i-r_ik_i \bmod \varphi(p) \tag{5-25}$$

最后将签名消息$(m', (s_i, r_i))$发给签名收集者U_c。

(3) 单用户签名的验证。U_c收到U_i的签名消息后，计算：

$$R = \prod_{i=1}^{n} r_i^{r_i} \bmod p \tag{5-26}$$

通过验证方程：

$$r_i^{r_i} g^{s_i} = y_i^{m'+R} \bmod p \tag{5-27}$$

来验证U_i签名的有效性。实际上，因为

$$s_i + r_i k_i = (R + m')x_i \bmod \varphi(p)$$

所以

$$r_i^{r_i} g^{s_i} = g^{k_i r_i + s_i} = g^{(m'+R)x_i \bmod \varphi(p)} = (g^{x_i})^{(m'+R)+\lambda\varphi(p)}$$
$$= y^{m'+R} y^{\lambda\varphi(p)} \bmod p = y^{m'+R} \bmod p$$

只有各U_i的签名均有效，才能计算多重签名：

$$S = s_1 + s_2 + \cdots + s_n \bmod p - 1$$

否则就终止签名。(R, S)即为多重签名。

(4) 多重签名的验证。验证者U_V验证$\mathrm{Ver}(m, r, s) = \mathrm{TURE}$的方法是：

$$Rg^{s} = \prod_{i=1}^{n} y_i^{m'+R} \tag{5-28}$$

理由是：

$$Rg^{S} = \prod_{i=1}^{n} r_i^{r_i} \cdot g^{(s_1+s_2+\cdots+s_n) \bmod (p-1)} = \prod_{i=1}^{n} g^{(k_i r_i + s_i) \bmod (p-1)}$$
$$= \prod_{i=1}^{n} g^{(R+m')x_i \bmod (p-1)} = \prod_{i=1}^{n} y^{(R+m')} y^{\lambda(p-1)} \bmod p$$
$$= \prod_{i=1}^{n} y_i^{m'+R} \bmod p$$

5.1.7 代理数字签名

设A、B是两用户，他们的私钥和公钥分别为(x_A, y_A)和(x_B, y_B)。如果满足以下条件：

(1) A利用他的秘密密钥x_A计算出一个数σ，并将σ秘密交给B。

(2) 任何人，包括B试图求出x_A时，σ不会对他有任何帮助。

(3) B可以用σ和x_B生成一个新的签名密钥$\sigma_{A\to B}$，并做签名$s = \mathrm{Sig}(\sigma_{A\to B}, m)$。

(4) 存在一个公开的验证算法$\mathrm{Ver}_{A\to B}$，使其对任何$s \in S$和$m \in M$，都有：

$$\mathrm{Ver}_{A\to B}(y_A, s, m) = \mathrm{TURE} \Rightarrow \mathrm{Sig}(\sigma_{A\to B}, m)$$

式中：S是所有签名的集合；M是所有明文的集合。

也就是说，对任何消息m及它的签名s，均可用A的公钥来验证。

(5) 任何人在试图求出x_A、x_B、σ或$\sigma_{A\to B}$时，$\mathrm{Sig}(\sigma_{A\to B}, m)$都不会对他产生帮助。

那么就说，用户A将他的签名权力委托给了用户B，称A为B的原始签名人，B为A的代理签名人，σ为托管密钥，$\sigma_{A\to B}$为代理托管密钥，$\mathrm{Sig}(\sigma_{A\to B}, m)$是对消息$m$做的A的代理签名。能够产生代理签名的体制则称为代理签名系统。

下面介绍一种基于离散对数的代理签名体制。

系统参数

(M, A, K, S, V)是基于离散对数的数字签名体制，其参数 p、q、g 满足：p 是大素数，在 Z_p 中离散对数是难解的；q 是大素数，且 $q \mid p-1$；$g \in Z_p^*$ 是 Z_p 域中 q 次单位元根。用户 A 和 B 的私钥分别是(x_A, y_A)和(x_B, y_B)，其中：

$$y_A = g^{x_A} \bmod p, \quad y_B = g^{x_B} \bmod p \tag{5-29}$$

委托过程

A 随机选择一整数 $k \in Z_p^*$，计算：

$$r = g^k \bmod p, \quad \sigma = x_A + kr \bmod g \tag{5-30}$$

将(σ, r)秘密传给 B，B 收到后验证等式 $g^{\sigma} = y_A r^r \bmod p$ 是否成立。

代理签名的生成算法

对于待签消息 m，B 生成普通的数字签名：

$$s = \mathrm{Sig}(\sigma, m)$$

比如

$$s = (x_B + \sigma m) \bmod g \tag{5-31}$$

将(s, r)作为他代表 A 对消息 m 的数字签名，此即代理签名。

代理签名的验证算法

当代理签名的接收者收到消息 m 和代理签名(s, r)后，计算：

$$v = y_A r^r \bmod p \tag{5-32}$$

验证：

$$\mathrm{Ver}(y_A, (s, r)) = \mathrm{TURE} \Rightarrow \mathrm{Ver}(v, s, m) = \mathrm{TURE}$$

比如验证：

$$g^s = y_B v^m \bmod p \tag{5-33}$$

证明

$$v = y_A r^r = g^{x_A} g^{kr} = g^{x_A + kr} = g^{\sigma} \bmod p$$

表明 v 可作为密钥 σ 的验证公钥。由 σ 对 m 所作数字签名 $s = \mathrm{Sig}(\sigma, m)$就可由 $\mathrm{Ver}(v, s) = m$ 来校验。证明：

$$y_B v^m = g^{x_B} (g^{\sigma})^m = g^{x_B + \sigma m} = g^s \bmod p$$

代理数字签名的安全性如下：

(1) 基本不可伪造性：B 不能根据(σ, r)求出 x_A，就无法伪造 A 的普通签名。

(2) 代理签名的不可伪造性：除 A、B 以外的人无法伪造有意义的代理签名。

(3) 代理签名的可区分性：代理签名由普通签名 s 和另一部分 r 组成，很容易与普通签名区分开来。如果同一人委托有两个代理人签名 A 的(s, r)和 C 的(s', r_2)，则由 $r \neq r_2$ 也很容易加以区分。

(4) 不可抵赖性：A 自然不能抵赖他传给 B 的(σ, r)，而 B 却可以说是 A 代理 B 而不是 B 代理 A，所以 B 必须是 A 的充分信任者。

(5) 身份证实性：A 见到一个代理签名(s, r)时，可以由 r 知道是 B 代理的。

(6) 可注销性：A 可以注销 B 的代理权，只需向大家声明 r 作废就可以。

5.2 单向散列(*Hash*)函数

5.2.1 概述

单向散列函数(Hash 函数)也叫杂凑函数，在上段所讲的数字签名中已多次出现过。它实际上是一种特殊的压缩算法，它把任意长度的消息 m 压缩成一定长度(如 128 bit、160 bit)的代码串(称之为消息摘要)。这种算法 $h(m)$ 应当保证原消息的每一符号与压缩结果相关联，以至于任意改变原消息的一个符号时将导致其信息摘要一半以上的 bit 变化。它还要求：

(1) $h(m)$ 应能对任意长度的消息 m 做计算。

(2) 计算 $h(m)$ 的值 h 是容易的，而由 h 计算 m 是不可行的(单向性)。

(3) 对于算法 $h(m)$，要找出两个不同消息 m_1 和 m_2 有相同的摘要值：

$$h(m_1) = h(m_2) \tag{5-34}$$

是非常困难的(或者说是不可行的)。

因此，$h(m)$ 可作为 m 的“指纹”或“缩影”方便地保存。当原文档 m 被传输或转存后，可再次计算其摘要值，比较是否变化(因为摘要值“放大”了文档的差别)，以判断原文档是否被有意或无意改动过。可见，单向散列函数最基本的功能是对文件做“完整性”检验。

其次，如果在计算摘要值的时候，必须利用发信人(作者)的私钥，则这个摘要值也就有了数字签名的功效。这样的签名短小方便，便于附于文后传输。因此摘要算法常常以多种方式与数字签名相结合使用。

下面介绍单向散列函数的常用构造方法。

5.2.2 用分组加密函数构造散列函数

第 2 章讲的分组加密算法，如 DES，它将明文 m 分成长为 64 bit 的许多小段 $m_1m_2\cdots m_n$，使用 64 bit 的密钥，一段一段地加密，联结起来就是密文。

现在可以把 m_1 加密得到的 64 bit 的密文段 c_1 作为密钥，去对 m_2 加密，用得到的结果 c_2 作为密钥，再对 m_3 加密，如此下去，直到最后得到 m_n 的密文 c_n，则 c_n 就是所求得的 64 bit 的信息摘要。

1. Rabin 方案

Rabin 方法加密产生信息摘要的方案如图 5.3 所示。

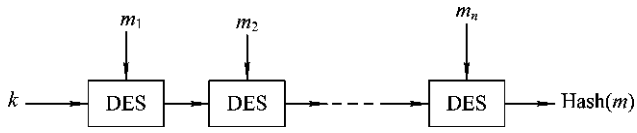

图 5.3 Rabin 方法加密产生信息摘要的方案

这种反复迭代计算，还可有多种不同方式，如利用分组链接方案、密文链接方案和密

钥链接方案加密产生信息摘要，分别如图 5.4～5.6 所示。

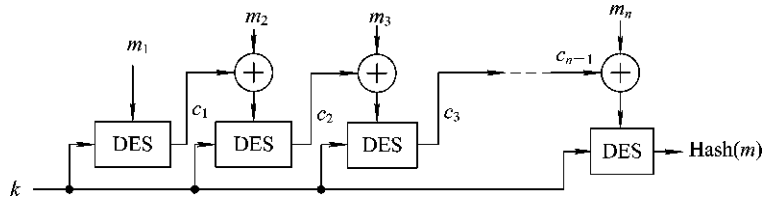

图 5.4　分组链接产生信息摘要的方案

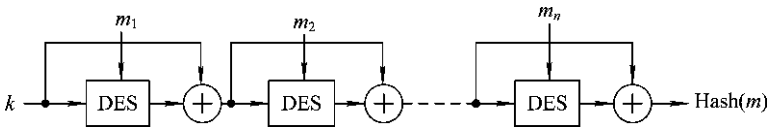

图 5.5　密文链接产生信息摘要的方案

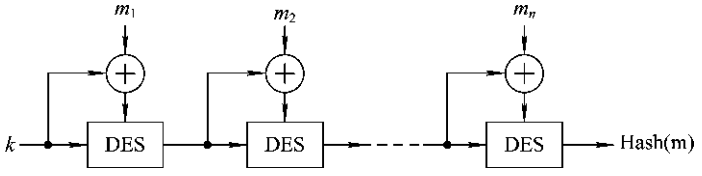

图 5.6　密钥链接产生信息摘要的方案

分组链接方案中，每次 c_i 与 m_{i+1} 按位模 2 加后，送入加密器进行加密。

5.2.3　用候选单向函数构造散列函数

(1) 若 $f=f(n, e)$ 为 RSA 函数

令
$$h_0=IV \quad (\text{初始值}) \tag{5-35}$$

则
$$h_i=(m_i+h_{i-1})^e \bmod n \qquad i=1, 2, 3, \cdots, t \tag{5-36}$$

(2) 若 $f=f(p, g)$ 为离散对数函数

令
$$h_0=IV \quad (\text{初始值}) \tag{5-37}$$

则
$$h_i=g^{(h_{i-1}+m_i)} \bmod p \qquad i=1, 2, 3, \cdots, t \tag{5-38}$$

总之，任何一种单向函数，迭代使用时都可构造 $h(m)$ 函数。

5.2.4　软件算法 MD4

MD4 算法是一种通过计算机程序将任意长消息压缩为 128 bit 消息摘要的算法。

首先将明文 x 按 512 bit 分段，最后不足 512 bit 的部分，扣除末尾 64 bit 后添一个 1 和 d 个 0，凑成 448 bit，再把这 64 bit 加在末尾，使之也成为 512 bit 的一段，共有 N 段。软件依次处理各个段落，每次把 512 bit 的一个段落分组成 32 bit 的 16 小段，装入数组：

$$M=M[0], M[1], \cdots, M[15] \tag{5-39}$$

(1) 给四个寄存器 A、B、C、D 赋初值，初值的十六进制表达为

$$A=67452301, B=efcdab89, C=98badcfe, D=10325476 \tag{5-40}$$

每符号 4 bit，A、B、C、D 都是 32 bit 的寄存器，正好可存数组的一个单元。

(2) 将 A、B、C、D 的数据拷贝到另外四个寄存器 AA、BB、CC、DD 中备用，而 A、

B、C、D 作为计算过程中的变量使用。所用到的运算符号说明如下：

$x \wedge y$ 表示按位逻辑与运算；$x \vee y$ 表示按位逻辑或运算

$x \oplus y$ 表示按位逻辑异或运算；$\overline{x}$ 表示按位逻辑非运算

$x+y$ 表示整数的模 2^{32} 加法；$x<<s$ 表示 x 循环左移 s 位($0<s<31$)

(3) for $i=0$ to$(N-1)$do(循环处理各个 512 bit 的段落)

(4) for $j=0$ to 15 do(循环处理 16 个 32 bit 数组单元)

$X[j] = M[j]$ (一次将一个 32 bit 装入数组 $X[j]$中)

(5) 执行第一轮：for $k=0$ to 3 do

$$A=\{A+f(B, C, D)+X[4k]\}<<3$$
$$D=\{D+f(A, B, C)+X[4k+1]\}<<7$$
$$C=\{C+f(D, A, B)+X[4k+2]\}<<11$$
$$B=\{B+f(C, D, A)+X[4k+3]\}<<19$$

其中：
$$f(x, y, z)=(x \wedge y) \vee (\overline{x} \wedge z) \tag{5-41}$$

(6) 执行第二轮：for $k=0$ to 3 do

$$A=\{A+g(B, C, D)+X[k]+5a827999\}<<3$$
$$D=\{D+g(A, B, C)+X[4+k]+5a827999\}<<5$$
$$C=\{C+g(D, A, B)+X[8+k]+5a827999\}<<9$$
$$B=\{B+g(C, D, A)+X[12+k]+5a827999\}<<13$$

其中：
$$g(x, y, z)=(x \wedge y) \vee (x \wedge z) \vee (y \wedge z) \tag{5-42}$$

(7) 执行第三轮：for $k=0$ to 3 do

$$A=\{A+h(B, C, D)+X[0]+6ed9eba1\}<<3$$
$$D=\{D+h(A, B, C)+X[8]+6ed9eba1\}<<9$$
$$C=\{C+h(D, A, B)+X[4]+6ed9eba1\}<<11$$
$$B=\{B+h(C, D, A)+X[12]+6ed9eba1\}<<15$$
$$A=\{A+h(B, C, D)+X[2]+6ed9eba1\}<<3$$
$$D=\{D+h(A, B, C)+X[10]+6ed9eba1\}<<9$$
$$C=\{C+h(D, A, B)+X[6]+6ed9eba1\}<<11$$
$$B=\{B+h(C, D, A)+X[14]+6ed9eba1\}<<15$$
$$A=\{A+h(B, C, D)+X[1]+6ed9eba1\}<<3$$
$$D=\{D+h(A, B, C)+X[9]+6ed9eba1\}<<9$$
$$C=\{C+h(D, A, B)+X[5]+6ed9eba1\}<<11$$
$$B=\{B+h(C, D, A)+X[13]+6ed9eba1\}<<15$$
$$A=\{A+h(B, C, D)+X[3]+6ed9eba1\}<<3$$
$$D=\{D+h(A, B, C)+X[11]+6ed9eba1\}<<9$$
$$C=\{C+h(D, A, B)+X[7]+6ed9eba1\}<<11$$
$$B=\{B+h(C, D, A)+X[15]+6ed9eba1\}<<15$$

其中：
$$h(x, y, z)=x \oplus y \oplus z \tag{5-43}$$

(8) 再令

$$AA = A + AA,\ BB = B + BB,\ CC = C + CC,\ DD = D + DD \tag{5-44}$$

回到(3)，处理下一个 512 bit 段落，直至全部。

(9) 最后将四个寄存器 AA、BB、CC、DD 中的 4×32 bit 链接，即得到 128 bit 的消息摘要。

5.2.5 MD5 算法

MD5 是对 MD4 的改进，消息分组方式不变。但初始值变为

$$A = 0x01234567,\ B = 0x89abcdef$$
$$C = 0xfedcba98,\ D = 0x76543210 \tag{5-45}$$

它用到的四个非线性函数是：

$$F(x, y, z) = x(\wedge y) \vee (\bar{x} \wedge z),\quad G(x, y, z) = (x \wedge z) \vee (y \wedge \bar{z})$$
$$H(x, y, z) = x \oplus y \oplus z,\quad I(x, y, z) = y \oplus (x \vee \bar{z}) \tag{5-46}$$

并且引入以下四个符号表示四种运算过程，各自引用不同的非线性函数：

$$FF(a, b, c, d, M_j, s, t_i) \text{ 表示 } a = b + \{[a + F(b, c, d)] + M_j + t_i\} << s$$
$$GG(a, b, c, d, M_j, s, t_i) \text{ 表示 } a = b + \{[a + G(b, c, d)] + M_j + t_i\} << s$$
$$HH(a, b, c, d, M_j, s, t_i) \text{ 表示 } a = b + \{[a + H(b, c, d)] + M_j + t_i\} << s$$
$$II(a, b, c, d, M_j, s, t_i) \text{ 表示 } a = b + \{[a + I(b, c, d)] + M_j + t_i\} << s \tag{5-47}$$

512 bit 的信息段被放入 16 个数组单元 M_j 中，各 32 bit，进行 4 轮加密，每轮循环 16 次，每次分别对一个数组单元进行，共 64 次。各次处理过程的程序全都相同，都是在 4 个变量中进行运算，先把其中 3 个变量进行非线性运算，然后加上第 4 个变量，再与一个数组单元内的被处理信息和一个常数 t_i 相加，之后循环左移 s 位，最后加上 3 个变量中的第一个，将结果赋予第 4 个变量。而 64 次处理过程的区别在于：A、B、C、D 四个变量与 a、b、c、d 四个形参的对应关系不同，每次引用的信息数组单元 M_j 不同，所加常数 t_i 不同，循环左移位数 s 不同，4 大轮所用的非线性运算函数也不同。

第一轮是：

$FF(A, B, C, D, X_0, 7, 0xd76aa478)$
$FF(D, A, B, C, X_1, 12, 0xe8c7b756)$
$FF(C, D, A, B, X_2, 17, 0x242070db)$
$FF(B, C, D, A, X_3, 22, 0xc1bdceee)$
$FF(A, B, C, D, X_4, 7, 0xf57c0faf)$
$FF(D, A, B, C, X_5, 12, 0x4787c62a)$
$FF(C, D, A, B, X_6, 17, 0x48304613)$
$FF(B, C, D, A, X_7, 22, 0xfd469501)$
$FF(A, B, C, D, X_8, 7, 0x698098d8)$
$FF(D, A, B, C, X_9, 12, 0x8b44f7af)$
$FF(C, D, A, B, X_{10}, 17, 0xffff5bb1)$
$FF(B, C, D, A, X_{11}, 22, 0x895cd7be)$
$FF(A, B, C, D, X_{12}, 7, 0x6b901122)$
$FF(D, A, B, C, X_{13}, 12, 0xfd987193)$
$FF(C, D, A, B, X_{14}, 17, 0xa679438e)$

$FF(B, C, D, A, X_{15}, 22, 0x49b40821)$

第二轮是：

$GG(A, B, C, D, X_1, 5, 0x561e2562)$
$GG(D, A, B, C, X_6, 9, 0xc040b340)$
$GG(C, D, A, B, X_{11}, 14, 0x265e5a51)$
$GG(B, C, D, A, X_0, 20, 0xe96607aa)$
$GG(A, B, C, D, X_5, 5, 0xd62f105d)$
$GG(D, A, B, C, X_{10}, 9, 0x02441053)$
$GG(C, D, A, B, X_{15}, 14, 0xd8a1e681)$
$GG(B, C, D, A, X_4, 20, 0xe7d3fbc8)$
$GG(A, B, C, D, X_9, 5, 0x21e1cde6)$
$GG(D, A, B, C, X_{14}, 9, 0xc33707d6)$
$GG(C, D, A, B, X_3, 14, 0xf4d50d87)$
$GG(B, C, D, A, X_8, 20, 0x455a14ed)$
$GG(A, B, C, D, X_{13}, 5, 0xa9e3e905)$
$GG(D, A, B, C, X_2, 9, 0xfcefa3f8)$
$GG(C, D, A, B, X_7, 14, 0x676f02d9)$
$GG(B, C, D, A, X_{12}, 20, 0x8d2a4c8a)$

第三轮是：

$HH(A, B, C, D, X_5, 4, 0xfffa3942)$
$HH(D, A, B, C, X_8, 11, 0x87715681)$
$HH(C, D, A, B, X_{11}, 16, 0x6d9d6112)$
$HH(B, C, D, A, X_{14}, 23, 0xfde5380c)$
$HH(A, B, C, D, X_1, 4, 0xa4beea44)$
$HH(D, A, B, C, X_4, 11, 0x4bdecfa9)$
$HH(C, D, A, B, X_7, 16, 0xf6bb4b60)$
$HH(B, C, D, A, X_{10}, 23, 0xbebfc70)$
$HH(A, B, C, D, X_{13}, 4, 0x289b7ec6)$
$HH(D, A, B, C, X_0, 11, 0xeaa107fa)$
$HH(C, D, A, B, X_3, 16, 0xd4ef3085)$
$HH(B, C, D, A, X_6, 23, 0x04881d05)$
$HH(A, B, C, D, X_9, 4, 0xd9d4d039)$
$HH(D, A, B, C, X_{12}, 11, 0xe6db99e5)$
$HH(C, D, A, B, X_{15}, 16, 0x1fa27cf8)$
$HH(B, C, D, A, X_2, 23, 0xc4ac5665)$

第四轮是：

$II(A, B, C, D, X_0, 6, 0xf4292244)$
$II(D, A, B, C, X_7, 10, 0x432aff97)$
$II(C, D, A, B, X_{14}, 15, 0xab9423a7)$
$II(B, C, D, A, X_5, 21, 0xfc93a039)$
$II(A, B, C, D, X_{12}, 6, 0x655b59c3)$
$II(D, A, B, C, X_3, 10, 0x8f0ccc92)$

$II(C, D, A, B, X_{10}, 15, 0xffeff47d)$
$II(B, C, D, A, X_1, 21, 0x85845dd1)$
$II(A, B, C, D, X_8, 6, 0x6fa87e4f)$
$II(D, A, B, C, X_{15}, 10, 0xfe2ce6e0)$
$II(C, D, A, B, X_6, 15, 0xa3014314)$
$II(B, C, D, A, X_{13}, 21, 0x4e0811a1)$
$II(A, B, C, D, X_4, 6, 0xf7537e82)$
$II(D, A, B, C, X_{11}, 10, 0xbd3af235)$
$II(C, D, A, B, X_2, 15, 0x2ad7d2bb)$
$II(B, C, D, A, X_9, 21, 0xeb868391)$

在第 i 步中，t_i 是 $2^{32}\times\text{abs}[\sin(i)]$的整数部分，$i$ 的单位是弧度。每 512 bit 处理完后与上一轮结果相加(同 MD4)，最后是 A、B、C、D 的级联。

5.2.6 对散列函数的攻击和安全性

1. 生日攻击

生日攻击指攻击者找到了与明文 m 有相同散列值的另一个不同的明文 m'。之所以叫这样一个名称，是因为它雷同于所谓生日问题：问一个班级至少有多少个学生，才能使两个学生同一天生日的概率不小于 1/2？

首先任意指定一位学生报出自己的生日，在其余的学生中，第 1 位学生的生日不在这一天的概率是 $364/365=1-1/365$，第 2 位学生的生日不在这两天的概率是 $363/365=1-2/365$ …… 第 j 位学生的生日不在已被前面学生占用的 j 天的概率是 $1-j/365$，因此，除了所有生日不相同的情况之外，就是出现两个或更多学生生日在同一天的概率：

$$p=1-\prod_{j=1}^{J-1}\left(1-\frac{j}{N}\right) \tag{5-48}$$

式中，$N=365$ 是一年的天数，J 是班级里学生总数，p 是出现两人或多人生日相同的概率。

由于：

$$e^{-x}=1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\cdots$$

当 x 较小时有 $1-x\approx e^{-x}$，因此，当 $J<<N$ 时，有

$$p\approx 1-\prod_{j=1}^{J-1}e^{-\frac{j}{N}}=1-e^{\sum_{j=1}^{J-1}-\frac{j}{N}}=1-e^{-\frac{J(J-1)}{2N}}$$

解得：

$$J-J^2\approx 2N\ln(1-p)$$

忽略一次项 J，得到：

$$J\approx\sqrt{(-2\ln(1-p))N} \tag{5-49}$$

当 $p=0.5$ 时，$J\approx 22.49$。这个答案是出乎意料的，仅仅 23 人就会有 50%的概率出现两人生日在同一天！

类似的情况也会发生在电话号码问题上。设某城市采用 7 位号码，共有 $N=10^7$ 个号码可选择，如果每个用户都能不受限制地随机选择号码，不难算出，只需 $J=3723$ 部电话，出现重号的概率就会达到 50%。

现在回到散列函数。n bit 长的散列函数共有 $N=2^n$ 个不同的组合，两个人生日相同在这里就是两个不同明文 m 和 m' 有相同的摘要值，求学生数目 J 就是求明文数目 J，问 J 多大时就有概率为 p 的可能出现两摘要值相同的情况？由式(5-49)知明文数 J 正比于 $\sqrt{N}$，即正比于 $2^{\frac{n}{2}}$。

显然，对于 $n=64$ 的散列函数(比如用 DES 构造)，安全性是差了一些。安全的 Hash 函数规定 $n=160$ bit 就是为了抵抗生日攻击。

2. 中间相迂攻击

中间相迂攻击是针对分组链接结构的散列函数进行的。攻击者随意选出若干消息分组，将它们分为两部分，一部分以初值 IV 开始迭代到中间某一步为止，得到 J_1 个输出；第二部分以摘要值 $h(x)$ 出发，用逆函数反向迭代到中间某一步为止，得到 J_2 个输出；然后比较这两部分输出，若能找到一对输出相等，则可得到一对“碰撞消息”。

第一部分按

$$h_0 = IV,\ h_i = E(x_i' + h_{i-1},\ k) \qquad i = 1,\ 2,\ \cdots,\ \frac{t}{2}$$

进行迭代(不妨假设 t 为偶数)。

第二部分按

$$h_t' = h(x),\ h_{i-1}' = D(h_i',\ k) + x_i' \qquad i = t,\ \cdots,\ \frac{t}{2}+1$$

进行迭代，其中 E 为加密函数，D 为解密函数。即

$$h_{i-1}' = D(h_i',\ k) + x_i'$$

是

$$h_i = E(x_i' + h_{i-1},\ k)$$

的逆函数。

因此两部分中若有一对相等，即 $h_{t/2} = h_{t/2}'$，则可将对应的消息链成一个消息：$x' = x_1'x_2'\cdots x_t'$，使 $h(x') = h(x)$。

由于 x' 是攻击者随机选的，因此可选 $x' = x$。这样就得到一对碰撞消息。为了找到这对碰撞消息，所需要选出的消息总数 $J = J_1 + J_2$。这时 J 不仅与分组加密函数的分组长度有关，也与函数的性质有关。

实际上，攻击的第一部分是选择明文攻击，第二部分是选择密文攻击，为了抵抗选择明/密文攻击，分组加密的分组长度至少应 128 bit。

防止中间相迂攻击的另一方法是避免采用可逆的加解密函数。

3. 修正分组攻击

修正分组攻击即伪造消息的一个分组，通常是最后那个分组，目的是要修正杂凑值，企图使它等于某个期望的值，以达到伪造签名的目的。

5.3 身份识别[4]

5.3.1 概述

识别(identification)和验证(authentication)是不同的。数字签名属于身份验证，当通

信双方或一方需要被验证时，通常存在一些承载信息在通信双方之间交换。而识别是对一个用户身份的实时证明，它不需要交换承载信息。

身份识别广泛地应用在访问控制中，根据身份识别决定允许或拒绝某用户访问相应的资源，如出纳机、登录计算机、识别入网的移动电话等。

识别是基于非时变的口令或密码的。这样的系统，用户和识别者都知道口令或密码，识别者就有可能冒充用户。实际上，无需让识别者知道秘密口令，只要让它有能力区别有效口令和无效口令即可。一种改进的方法是通过单向函数将用户输入的口令加密，识别者的数据库里只存储着每个用户的口令加密值。

这样改进的系统也未必能十分安全，因为它无法抵制字典式攻击。其次，当用户输入口令时也难免被人窥视到。一个安全的识别协议至少应该在证明自己身份时不泄露秘密。这可能吗？完全可能！下面所举的 GSM 身份识别就是这样的强识别方案。

(1) A 通过密码和用户名向 B 登记(A 为移动手机用户，B 为服务商网络)。

(2) B 发给 A 一个随机号码(询问)。

(3) A 用自己的私钥对这个随机号码加密后回答 B。

(4) B 验证回答正确，表明 A 确实拥有密钥。

窃听者不能重发这个响应，因为当他联系 B 时，询问已经不同了。而密钥在整个过程中并未泄露，窃听者无法得到。

当然，这个方案是建立在 A、B 双方互相信任的基础之上的。然而很多情况下，双方未必能互相信任，甚至还可能是敌对的，所以更加安全的识别方案应该是不需要双方共享秘密的方案。其识别协议至少应包含以下两条：

(1) 证明者 A 能向验证者 B 证明他的确是 A。

(2) 在证明者 A 向验证者 B 证明其身份后，B 并未获得任何关于 A 的有用信息，他无法模仿 A 向第三者证明自己是 A(冒充 A)。

这个协议要求 A 在证明自己身份的同时没有向 B 泄露任何识别信息。

下面介绍几种这样的识别方案。

5.3.2 Schnorr 身份识别方案

1991 年 Schnorr 提出一种计算量、通信量都很少的识别方案，特别适用于在智能卡(Smart Card)上使用。

Schnorr 方案需要一个可信任的验证中心 TA，TA 选择以下参数：

(1) p、q 是两个大素数，且 $q|(p-1)$，q 至少为 140 bit，而 p 至少 512 bit。

(2) $\alpha \in Z_p^*$ 为 q 阶元(诸如可取 $\alpha = g^{(p-1)/g}$，g 为 Z_p 的本原元。)

(3) $h(x)$是输出为 t bit 的单向函数，t 是一个安全参数($t=40$ bit 或 $t=72$ bit)。

(4) 每个用户暗选私有密钥 s：$s \in [1, q-1]$；算出公开密钥 v：$v = \alpha^{-s} \bmod p$。

(5) 每个用户还必须到 TA 注册其公开密钥，TA 验证后对每位用户指定一识别名 I。I 中包括用户的姓名、性别、生日、职业、电话号码、指纹信息甚至 DNA 等识别信息。TA 对 $h(I, v)$予以签名，以备将来验证身份时查用。

身份识别协议如下：

(1) 用户 A 将自己的识别名 I 和公钥 v 交给验证者 B。

(2) B向TA索取签名以验证A提供的信息无误。

(3) 用户A任意选择一整数r，$1 \gg r \gg q-1$，计算$X=\alpha^r \bmod p$发给B。

(4) 验证者B任选取一整数$e \in [1, 2^t]$送回A。

(5) 用户A计算$y=r+se \bmod q$，并送回B。

(6) 验证者B验证$\alpha^y \times v^e \bmod p$，判断其是否等于$X$。

实际上，若用户诚实，则

$$\alpha^y \times v^e = \alpha^{r+se} \times (\alpha^{-s})^e = \alpha^r \bmod p = X$$

在此过程中，并不需要TA的介入。A并未向B暴露自己的私钥，而B完全可以相信A拥有私钥s，否则无法得到$\alpha^y \times v^e = X$的结果。这个方案本质上不需要事先分享秘密。

针对一般交互式用户身份证明协议，有三方面要求：

(1) 完全性(Completeness)：对于诚实的合法用户和验证者，验证成功的概率几乎百分之百。

(2) 健全性或合理性(Soundness)：当不知道用户信息和私钥时，验证通过的概率几乎为零。

(3) 隐蔽性(Witness hiding)：若用户是诚实的，无论协议执行多少次，任何人(包括验证者)都无法推知用户的密钥，无法冒充这个用户。

然而，一个满足完全性和合理性的协议，未必是安全的。比如A可以把私钥有意泄露给Oscar，从而Oscar能够模仿A，使协议变得不安全。但完全性和合理性却不受影响。

5.3.3 Okamoto身份识别方案

Okamoto改进了Schnorr方案，使方案具有了安全性。然而计算速度和有效性却不如Schnorr。

系统参数

TA选择两个大素数p和q，在Z_p中选择两个阶为q的元素α_1、α_2，公开p、q、α_1、α_2，但对系统所有的参加者包括A均保密$c=\log_{\alpha_1}\alpha_2$(即保密它们之间的关系$\alpha_1^c=\alpha_2$)。我们假定任何人，即使A与Oscar合伙计算$c$也是不可行的。

另外，TA选择一个签名方案和Hash函数。

颁布一个证书的协议为

(1) TA建立A的身份，并颁布一个识别串ID(A)；

(2) A秘密地选择了两个随机指数m_1、m_2($0 \leqslant m_1, m_2 \leqslant q-1$)，并计算：

$$v = \alpha_1^{-m_1}\alpha_2^{-m_2} \bmod p \tag{5-50}$$

同时将v发给TA。

(3) TA对(ID，v)签名：$s=\mathrm{Sig}_{\mathrm{TA}}(\mathrm{ID}, v)$，同时TA将证书$C(\mathrm{A})=(\mathrm{ID}(\mathrm{A}), v, s)$发送给A。

Okamoto的识别协议为

(1) A随机选择两个数r_1、r_2($0 \leqslant r_1, r_2 \leqslant q-1$)，并计算：

$$X = \alpha_1^{r_1}\alpha_2^{r_2} \bmod p \tag{5-51}$$

(2) A将他的证书$C(\mathrm{A})=(\mathrm{ID}(\mathrm{A}), v, s)$和$X$发给B。

(3) B通过检测$\mathrm{Ver}_{\mathrm{TA}}(\mathrm{ID}, v, s)=\mathrm{TURE}$来验证TA的签名。

(4) B 随机地选择一个数 e，$1 \leqslant e \leqslant 2^t$，$t$ 为安全参数，将 e 发给 A。

(5) A 计算：

$$y_1 = (r_1 + m_1 e) \bmod q,\ y_2 = (r_2 + m_2 e) \bmod q \tag{5-52}$$

并将 y_1、y_2 发给 B。

(6) B 验证：

$$X = \alpha_1^{y_1} \alpha_2^{y_2} v^e \bmod p \tag{5-53}$$

实际上，因为

$$\alpha_1^{y_1} \alpha_2^{y_2} v^e = \alpha_1^{r_1+m_1 e} \alpha_2^{r_2+m_2 e} (\alpha_1^{-m_1} \alpha_2^{-m_2})^e = \alpha_1^{r_1} \alpha_2^{r_2} = X$$

所以在离散对数 $c = \log_{\alpha_1} \alpha_2$ 是不可行的情况下，该方案是安全的。

5.3.4 Guillou-Quisguater 身份识别方案

该方案是基于 RSA 体制的安全识别方案。

TA 选择两个大素数 p 和 q，形成 $n = pq$，保留 p 和 q，公开 n；TA 选择一个大素数 b，一般 b 是一个长为 40 bit 的素数；另外，TA 选择了一个签名方案和 Hash 函数。

TA 向证明者 A 颁布一个证书的协议如下：

(1) TA 建立 A 的身份，并颁布一个识别串 ID(A)给 A。

(2) A 秘密地选择了一个整数 $m(0 \leqslant m \leqslant n-1)$，A 计算 $v = (m^{-1})^b \bmod n$，并将 v 发给 TA。

(3) TA 对(ID，v)签名：$s = \mathrm{SIG}_{\mathrm{TA}}(\mathrm{ID}, v)$，同时 TA 将证书 $C(\mathrm{A}) = (\mathrm{ID}(\mathrm{A}), v, s)$发送给 A。

证明者 A 向验证者 B 证明身份的识别协议为

(1) A 随机选择一个整数 $r(0 \leqslant r \leqslant n-1)$，并计算

$$X = r^b \bmod n \tag{5-54}$$

(2) A 将他的证书 $C(\mathrm{A}) = (\mathrm{ID}(\mathrm{A}), v, s)$和 X 发给 B。

(3) B 通过检测 $\mathrm{Ver}_{\mathrm{TA}}(\mathrm{ID}, v, s) = \mathrm{TURE}$ 来验证 TA 的签名。

(4) B 随机地选择一个整数 $e(0 \leqslant e \leqslant b-1)$，并将 e 发给 A。

(5) A 计算：

$$y = rm^e \bmod n \tag{5-55}$$

并将 y 发给 B。

(6) B 验证

$$X = v^e y^b \bmod n \tag{5-56}$$

以此来决定 A 的身份。

当 A 合法时，自然有

$$v^e y^b = (m^{-1})^{be} (rm^e)^b = r^b \bmod n = X$$

5.3.5 Shamir 基于身份的密码方案

Shamir 1984 年提出基于 Smart 卡(智能卡)的一种密钥方案，它能使网上任何一对用户无需交换密钥，也无需保存公钥簿，中间过程无需第三方服务，即可安全地通信和相互验证签名。当然，这个 Smart 卡是每位用户入网时由可信任中心颁发的。该 Smart 卡包含

一个微处理器，一个I/O端口，一个RAM，一个带有秘密密钥的ROM和加解密消息产生签名的程序。用户的公钥，就是自己的姓名和网址。

当用户A想给用户B发送一条消息m时，就用自己的Smart卡中的秘密密钥对消息m签名，用B的名字和网址来加密签名和消息，连同自己的姓名、网址一起发送给B；当B收到后，他使用自己的Smart卡中的秘密密钥进行解密，最后用发信者的姓名、网址作为公钥来验证签名。

Smart卡成了用户身份的全权代表，因此，如果丢失、复制或未授权使用，将造成严重后果。为此，可信任中心在颁发Smart卡时，并不是将用户的私钥完整地写入卡中，而是写入一个生成函数，只要用户输入一个设定的"种子"密钥k，就能根据用户信息ID和k生成该用户的私钥来。而这个较短的k值就是用户应牢记心中的密钥，系统要求：

(1) 当知道种子k时，根据姓名、网址能够容易地求出秘密密钥。

(2) 从一对特定的公钥(姓名、网址)和私钥求出种子k是困难的。

有了这两个附加条件，可构造多种识别方案。

Guillou-Quisguate基于身份的识别方案就是这样一个方案。TA为用户A颁布一个数u，该数是A的身份ID串的函数，而无需为用户A颁布一个证书。参数选择同前面所讲的Guillou-Quisguate协议(见5.3.4节)。

TA向A颁布一个数u的协议如下：

(1) TA建立A的身份，并颁布一个识别串ID(A)；

(2) TA计算

$$u = (H(\mathrm{ID(A)})^{-1})^a \bmod n \tag{5-57}$$

并将u发给A。其中$H(\cdot)$是一个公开的Hash函数；a是TA的私钥；b是TA的公钥；a、b满足关系：

$$ab = 1 \bmod \varphi(n) \tag{5-58}$$

身份的识别协议为

(1) A随机选择一个整数$r(0 \leqslant r \leqslant n-1)$，并计算

$$X = r^b \bmod n \tag{5-59}$$

(2) A将他的ID(A)和X发给B。

(3) B计算

$$v = H(\mathrm{ID(A)}) \tag{5-60}$$

(4) B随机选择一个整数e，$0 \leqslant e \leqslant b-1$，并将$e$发给A。

(5) A计算

$$y = ru^e \bmod n \tag{5-61}$$

并将y发给B。

(6) B验证

$$X = v^e y^b \bmod n \tag{5-62}$$

是否成立。

实际上，这是因为

$$v^e y^b = \underbrace{H(\mathrm{ID(A)})^e}_{v} \cdot r^b \cdot \underbrace{(H(\mathrm{ID(A)})^{-1})^{aeb}}_{u} = r^b \bmod n$$

5.4 消息认证码(MAC)

5.4.1 消息认证

前三节讨论的数字签名与身份认证都是基于公开密钥的双钥制建立的。其实，认证技术并非双钥制所独有，单钥制下也完全可以进行认证。不仅如此，而且单钥制下的认证技术在通信与计算机网络中早已广泛应用，即使在双钥制备受青睐的今天，单钥制由于其加、解密运算简单方便、快捷省时而仍然具有广阔的市场。双钥制则因为运算速度慢与结构复杂等原因，往往只用于单钥制无能为力的地方，比如密钥交换与特殊签名等，并不能完全替代单钥制。消息认证码就是单钥制下的最基本的认证技术。

消息认证包含三方面含义：

(1) 对消息内容的认证。消息在传送过程中可能出现差错或者被有意篡改与伪造，通过完整性验证，可确保内容是真实合法的。

(2) 对消息来源的认证。攻击者可能伪装成发送者，发出假冒消息，可通过使用双方的共享密钥，确保发送者是其声称的合法用户。

(3) 对消息时效性的认证。消息在传送过程中可能被重传或延迟，可通过消息认证验证消息发送的先后顺序以及是否实时发送。

消息认证过程没有第三方参与，只在通信双方之间进行。消息认证的实质是：发送方通过双方协商的某种函数，以待发消息作为输入产生一个叫做消息认证码的认证信息，该信息值对于要保护的消息来说是唯一的，它与原始消息一起在公开信道上传送；接收方收到消息后，使用相同的函数对收到的消息再次计算，也得到一个认证信息码。如果两个认证信息码相同，则说明消息是合法可信的。这个做法与 Hash 函数完全类同，它用来验证消息的完整性。

典型的消息认证系统如图 5.7 所示。

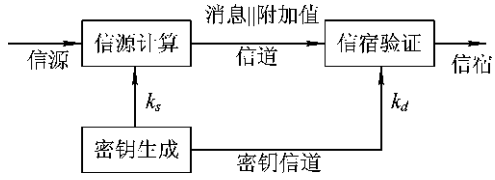

图 5.7 消息认证系统

图 5.7 中，消息的“附加值”就是消息认证码，它不仅应当与所发消息的每一字符紧密相关，还应当与发信人密钥密切有关，这样才能起到消息完整性认证与消息来源认证的双重功效。

那么消息认证与数字签名有何不同呢？双钥制中的数字签名是用发信人的私钥对消息或消息的摘要值加密的结果，它代表发信人的身份。而单钥制中的消息认证码则是用发信人与收信人所共有的密钥来处理消息的结果，它虽然能够证明消息不是第三者所为，但是不能抵制二人之间互相抵赖的情况，因此它只能用来作为认证消息的代码，而不是唯一代

表发信人凭据的签名。

5.4.2 消息认证码的算法[20~22]

消息认证是单钥制中广泛使用的消息认证方法，它通过用户的共享密钥 k 对变长消息 m 生成一个被称为消息认证码的固定大小的数据块，并附加到待发送消息之后。该算法就是 MAC(Message Authentication Code)函数，或者叫做密码校验和(Cryptographic Checksum)。

$$\mathrm{MAC} = C(k, m) \tag{5-63}$$

式中，C 代表 MAC 函数，是消息和密钥的公开函数，它类似于 Hash 函数，是不可逆的压缩函数。

通常，MAC 算法主要有两种类型：一种是基于分组密码的 MAC 算法；另一种是基于带密钥的 Hash 函数的 MAC 算法(HMAC)。二者的区别在于前者使用了与 Hash 函数相同的技术，而后者则直接调用现成的 Hash 函数。

1. CBC-MAC 算法

CBC-MAC 算法是基于 DES 的密文分组连接(CBC)方式。运算前，先对待认证消息进行 64 bit 长度的分组，得到 $m_i(i=1, 2, \cdots, N)$；最后的一个分组若不足 64 bit 则以 0 补齐。通过如图 5.8 所示的迭代算法得到压缩结果 C_N。最终的 MAC 可以是 64 bit 的 C_N，也可以是 C_N 最左边的 n 位，要求 $16 \leqslant n \leqslant 64$。

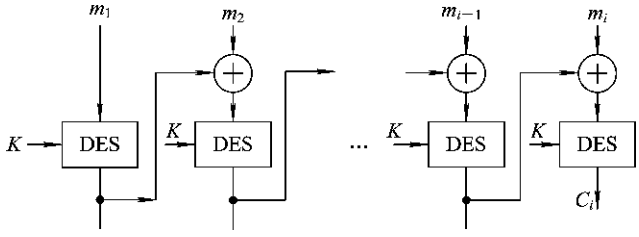

图 5.8 基于 DES 的 CBC-MAC 算法

2. HMAC 算法[20~22]

HMAC(Keyed-Hashing for Message Authentication Code)[23]是于 1996 年提出的一种基于 Hash 函数的消息认证方法，MD4、MD5 和 SHA-1 等均可应用于 HMAC。由于 Hash 函数具有效率高的优点，因此它的运行速度比基于分组密码的 MAC 更高。

HMAC 已被作为 ANSI、IETF、ISO 和 NIST 的标准，并被选为 IPSec 中实现 MAC 必须使用的方法，其他 Internet 协议，如安全套接协议 SSL 中也使用了 HMAC。

HMAC 算法使用密钥将任意长度的消息进行压缩。对于相同长度的消息，HMAC 算法的复杂性取决于执行底层 Hash 函数的次数。

系统参数

以 H 表示算法中嵌入的散列函数，K 为共享密钥。所嵌入的 Hash 函数可以不加修改地应用于 HMAC 中。在进行 HMAC 计算之前，首先将消息 M 分成长度为 b byte 的分组(一般 b=64)。IV 为初始值，ipad 和 opad 是计算中使用的两个固定的符号序列：

ipad = 00110110 (即 0x36)重复 $\frac{b}{8}$ 次后的序列

opad = 01011100(即 0x5C)重复 $\frac{b}{8}$ 次后的序列

算法流程

$S1$：若 length(K)=b，则转 $S4$

$S2$：若 length(K)>b，则 $K \leftarrow H(K)$，转 $S4$

$S3$：若 length(K)<b，则 $K \leftarrow (K \| 00\cdots0)$　　//添加 0 的个位数为 b−length(K)

$S4$：$S0 \leftarrow K \oplus$ ipad

$S5$：$M \leftarrow S0 \| M$

$S6$：$S0 \leftarrow H(M^{+}, IV)$

$S7$：$S1 \leftarrow K \oplus$opad

$S8$：$S \leftarrow S1 \| S0$

$S9$：$S \leftarrow H(S, IV)$

此过程可以简单描述为

$$\mathrm{HMAC}_K(M) = H[(K \oplus \mathrm{opad}) \| H[(K \oplus \mathrm{ipad}) \| M]] \tag{5-64}$$

需要说明的是，为了与 MAC 一致，可以对最后的输出 S 截取左边若干位作为消息鉴别码。考虑到避免 Hash 的生日攻击等安全因素，RFC2104 推荐截取长度不少于 Hash 输出长度的一半，且不少于 80 bit。如通过 MD5 实现的 HMAC－MD5－128 取输出的 128 bit 作为 MAC；通过 SHA－1 实现的 HMAC－SHA1－160 算法则使用 160 bit 作为输出结果。

5.4.3 MAC 的使用方式

消息认证码(MAC)是一种能够有效实现消息认证的技术，它可以将消息认证的功能与保密功能分离开来。MAC 在应用中主要有三种使用方式。

(1) 实现消息认证的功能：

① 发送方 A 计算 $\mathrm{MAC}_1 = C(k, m)$。

② A 将 MAC_1 连接在明文 m 之后，得到 $m \| \mathrm{MAC}_1$，并发送给 B。

③ 接收方 B 收到 $m \| \mathrm{MAC}_1$，提取 m。

④ B 使用共享密钥 k 计算 $\mathrm{MAC}_1' = C(k, m)$，如果 $\mathrm{MAC}_1 = \mathrm{MAC}_1'$，则可判定 m 未曾被修改过，并且该消息确实来自于 A。

(2) 实现对明文的消息认证和加密的功能：

① 发送方 A 计算 $\mathrm{MAC}_2 = C(k_1, m)$。

② A 将 MAC_2 连接在明文 m 之后，得到 $m \| \mathrm{MAC}_2$。

③ A 对 $m \| \mathrm{MAC}$ 通过共享密钥进行加密运算，即 $E_{k2}(m \| \mathrm{MAC}_2)$，并发送给 B。

④ B 收到后，首先进行解密运算，即 $m \| \mathrm{MAC}_2 = D_{k2}(E_{k2}(m \| \mathrm{MAC}_2))$，并提取 m。

⑤ B 利用共享密钥 k_1 计算 $\mathrm{MAC}_2' = C(k_1, m)$，如果 $\mathrm{MAC}_2 = \mathrm{MAC}_2'$，则可判定 m 未曾被修改过，并且该消息确实来自于 A。

(3) 实现对密文的消息认证和加密的功能：

① 发送方 A 通过共享密钥 k_2 对消息进行加密运算，即 $E_{k2}(m)$。

② A 通过共享密钥 k_1 计算 $\mathrm{MAC}_3 = C(k_1, E_{k2}(m))$。

③ A 将 MAC_3 连接在 $E_{k2}(m)$之后，得到 $E_{k2}(m) \| \mathrm{MAC}_3$，并发送给 B。

④ B收到后，提取 $E_{k2}(m)$。

⑤ B用密钥 k_1 计算 $\mathrm{MAC}_3'=C(k_1, E_{k2}(m))$。如果 $\mathrm{MAC}_3=\mathrm{MAC}_3'$，则可判定 $E_{k2}(m)$ 未曾被修改过，并且该消息确实来自于A。

⑥ B利用共享密钥 k_2 计算 $m=D_{k2}(E_{k2}(m))$，得到明文。

上述三种算法中，k、k_1 和 k_2 都是只有A和B拥有的共享密钥。

5.4.4 MAC算法的安全性

假设消息长度为 N，密钥长度为 k，MAC的长度为 n(通常建议 $k>n$)，根据MAC的基本原理，MAC是比明文短得多的数据块，因此有 $N\gg n$，也就是说，MAC函数的输入与输出是多对一的关系。在没有提供保密性的简单MAC中，攻击者试图找到密钥的主要方法是穷举法。在已知明文 m_1 及认证码 $\mathrm{MAC}_1=C(k_1, m_1)$的情况下，为了获得相应的密钥 k_1，攻击者需尝试 2^k 次，从而得到 2^k 个MAC。实际上不同的MAC只有 $2^n(2^k>2^n)$个，所以必然会发生不同密钥得到相同鉴别码MAC的情况。平均来说，可能有 $2^k/2^n$ 个密钥会产生相同的MAC，因此攻击者还需要对这些密钥再进一步筛选，分别计算 $2^k/2^{2n}$ 次、$2^k/2^{3n}$ 次 …… 共约 k/n 轮尝试，最终得到唯一正确的密钥。可见这种穷举法攻击难度很大。

为了确保安全性，在设计MAC函数时应满足如下要求：

(1) 已知 m_1 和 $C(k_1, m_1)$，要找出另一个不同的消息 m_2 在相同密钥下，与 m_1 有相同的MAC是不可行的。

(2) 对任意 $m_1\neq m_2$，找到 $C(k_1, m_1)=C(k_1, m_2)$的概率是 2^{-n}，n 是MAC的长度。

(3) 若 m_2 是 m_1 的某种已知变换，即 $m_2=f(m_1)$，则 $C(k_1, m_1)=C(k_1, m_2)$的概率也是 2^{-n}。

(4) HMAC的安全性不会低于所嵌入的Hash函数的安全性。

易知，上述第(2)条可以抵抗已知明文的穷举攻击；第(3)条则要求认证算法的安全性对消息的任意部分都应均匀，不应存在相对弱的部分。

习 题 5

1. 使用ElGamal数字签名算法，若 $p=2357$，$g=2$(2是 Z_p^* 的一个生成元)，私钥为 $x=1751$，随机数选取 $k=1529$。试对消息 $m=1463$ 进行签名，并对其验证。

2. 说明MD4和MD5的区别。MD5哪些方面更强于MD4？

3. 使用DSA数字签名，取 $q=101$，$p=78\times 101+1=7879$，$g=170$。假设私钥 $x=75$，那么 $y=g^x \bmod p=4567$。若选择随机数 k 为50，试对散列码1234签名，并给以验证。

4. 在ElGamal签名方案中，若发送方对于不同消息进行加密时使用相同的随机数 k，试分析接收方可以找到 k 并得到发送方的私钥 x 的可能性。

5. 当需要同时对消息进行签名和保密时，它们应该以何种顺序作用于消息？为什么？

6. 为什么可对消息的摘要进行数字签名？其作用是什么？

7. 考虑公钥加密算法构造杂凑函数，设算法是 RSA，将消息分组后用公钥加密第一个分组，加密结果与第二个分组异或后，再对其加密，一直进行下去。设一消息被分成两个分组 B_1 和 B_2，其杂凑值为 $H(B_1, B_2)=\mathrm{RSA}(\mathrm{RSA}(B_1)\oplus B_2)$。试证明对任一分组 C_1，可选 C_2，使得 $H(C_1, C_2)=H(B_1, B_2)$。再证明用这种攻击方法，可攻击上述公钥加密算法构造的杂凑函数。

8. 在一个四口之家，没有两个人的生日在同一个月的概率是多少？(假设所有的月份有相同的概率。)

9. 设 E_k 和 D_k 描述一个加密体制的加密和解密，其密钥为 K，因此若明文为 m，那么 $E_k(m)=c$ 是密文，并且 $D_k(c)=m$。如果 K 和 L 是两个密钥，那么很容易找到(并且存在)一个密钥 K_1，对于所有的 m 都满足 $E_L(E_k(m))=E_{k1}(m)$。假设已知一个明文密文对 (m_0, c_0)，则

(1) 说明怎样使用一个生日攻击得到密钥 K_0，使得 $E_{k0}(m_0)=c_0$；

(2) 若 K_1 和前面一样，存在但是很难找到，试说明怎样用中间迂回攻击找到一个解密函数(但是你可能找不到解密密钥)。

10. 一个电话公司职员用随机的 7 位数来分配电话号码。在一个有 10 000 部电话的城市，两个人分到同一个电话号码的概率是多少？有多少部电话时，号码相同的概率为 50%。

实践练习 5

实验题目：用 MD5 信息摘要进行数字签名的安全通信。

实验平台：Turbo C。

实验内容：用 MD5 信息摘要为任意明文进行数字签名，对附有签名的密件进行解密与认证。

实验步骤：

(1) 对任意长度明文进行分组，并转换成二进制码(如 ASCII 码)。

(2) 计算明文的 MD5 信息摘要。

(3) 借助 RSA 密钥体系用发信人私钥将信息摘要加密成数字签名。

(4) 将数字签名附于明文末尾。

(5) 收信人从信件末尾分离出数字签名，并用发信人公钥解译出信息摘要。

(6) 收信人再次使用 MD5 压缩明文，来验证文档的完整性。

(7) 收信人尝试稍许改变信件文字，则信息摘要便大不相同。

第 6 章　密钥管理和密码协议

密钥是现代密码体系中最关键的一些数据，对整个系统的安全起着至关重要的作用。在一个庞大的通信网络中，存在大量不同级别不同权限的用户，有着形形色色的密钥需要分配和认证，这无疑是一个非常复杂的问题，需要有效地加以管理。本章主要讨论密钥的产生、密钥的分配、密钥共享和密码协议等问题。最后作为实例讨论公钥基础设施 PKI。

6.1　密 钥 管 理[4]

6.1.1　概述

现代密码体系的一个基本概念是算法可以公开，而私有密钥则必须是保密的。不管算法多么强有力，一旦密钥丢失或出错，不但合法用户不能提取信息，而且可能导致非法用户窃取信息，甚至系统遭到破坏。

一个安全系统，不仅要阻止侵入者窃取密钥，还要避免密钥的未授权使用、有预谋的修改和其他形式的操作，并且希望当这些不利的情况发生时，系统应能及时察觉。

密钥管理应解决密钥自产生到最终销毁的整个过程中所涉及的各种问题，包括产生、装入、存储、备份、分配、更新、吊销和销毁。其中最棘手的是分配和存储。

1. 密钥的种类

(1) **基本密钥**(Base Key)

基本密钥也称初始密钥(Primary Key)，由用户自选，或由系统分配给用户，在较长时间内由该用户所专用，以 k_0 表示。

(2) **会话密钥**(Session Key)

会话密钥是两个通信终端在一次通话时所采用的密钥，由用户双方预先约定或动态地产生，以 k_1 表示。它的作用是使我们不必频繁地更换基本密钥，同时又能增加安全性。

(3) **密钥加密密钥**(Key Encrypting Key)

在交换会话密钥时，需要加密传送，用来加密会话密钥的密钥，就是密钥加密密钥，用 k_e 表示。一般各终端各有一把相互不同的 k_e 与主机交换会话密钥，而主机则不仅需要与各终端交换会话密钥，还要与其他主机之间交换密钥。

(4) **主机主密钥**(Host Master Key)

主机主密钥是对密钥加密密钥进行加密的密钥，用于主机给各用户分配 k_e，用 k_m 表示。

其他还有一些特殊用途的密钥，如用户选择密钥(Custom Option Key)、族密钥(Family Key)、算法更新密钥(Algorithm Changing Key)等。

2. 密钥的产生

靠人工产生和管理密钥的方法已被时代所淘汰，目前已有多种密钥产生器为大型系统提供各类密钥。

主机主密钥通常用诸如掷硬币、骰子或从随机数表中选数等随机方式产生，以保证密钥的随机性，避免可预测性。

密钥加密密钥可由机器自动产生，见图 6.1。常用的密钥产生器有随机比特发生器(如噪声二极管振荡器)或伪随机数发生器，也可由主密钥控制下的某种算法产生。

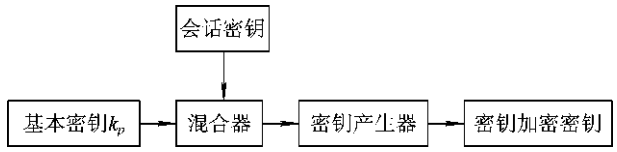

图 6.1 数据加密密钥的产生

会话密钥可通过某种加密算法动态地产生，如用初始密钥控制下的非线性移位寄存器或主密钥控制下的 DES 算法产生。

初始密钥的产生与主机主密钥或密钥加密密钥产生的方法相同。实践表明，增加密钥长度固然很必要，然而更重要的是它的产生算法和选择方法。有的人喜欢使用一些单词为密钥，为的是容易记忆，但是采用字典攻击常能奏效，按字典上的词汇一一进行尝试，这样 25%的密钥可被猜到。因此我们建议：

(1) 采用真正随机等概的序列为密钥，如掷硬币的结果。

(2) 避免使用特定算法的弱密钥。

(3) 采用揉搓和杂凑技术，将易记的词组句子经过单向函数处理变成伪随机数串。

3. 密钥的更换

密钥使用时间越长，泄露的机会就越大。所以密钥，特别是会话密钥要经常更换，使对手无法捉摸或难以搜索。密钥被更换的频率依据如下几点确定：

(1) 被保护数据的敏感性。

(2) 被保护数据的有效期。

(3) 密码算法的强度。

4. 密钥的存储

密钥的存储必须保障密钥的机密性、认证性和完整性。应防止密钥泄露和被篡改。

比较好的存储方式是将密钥存于磁条卡，或嵌入 ROM 芯片中，这样用户在使用时自己都不知道密钥是什么。还可将密钥分为两半，卡上和终端机中各一半。另外一个办法就是对密钥加密，使密钥永远不以未加密的方式出现在设备之外。

5. 密钥的销毁

不用的旧密钥必须销毁，否则可能造成危害。别人可能用它读取曾用它加密的文件，分析加密体系。对于写在纸上的密钥要用碎纸机粉碎，对于存在于介质上的密钥，要多次冲写。

6. 密钥的吊销

在密钥使用期内，发生丢失或其他不安全事件时，就需要从所使用的密钥集合中将其除去，并且通知系统防范。证书形式的公钥，可通过注销公钥证书的方式吊销。

6.1.2　密钥分配协议

密钥分配(Key Distribution)指的是这样一种机制：系统中某一个成员选择了一个秘密密钥，系统有能力将它安全地传送给另一个成员；设想有一个 n 用户的不安全网络，当某两位用户想保密通信时，先向可信任中心 TA 提出申请，TA 为他们选择一个会话密钥，通过安全信道(不在网络上进行)传给他们；这样就需要 n 条信道；若系统中两两都要通信，则每人须保存$(n-1)$个密钥，而 TA 则须保存 $n(n-1)$个密钥；显然当 n 较大时，代价是十分昂贵的。这称为 n^2 问题。

1. 对称系统的密钥分配

对称系统指通信双方使用同一把密钥的保密系统。这种系统又有在线和离线之分。前面所述的有 n 个安全通道的系统就是离线系统，而在线系统则不必保存那么多密钥。想利用网络通信的用户临时向 TA 申请一个会话密钥，每次都可以不同，以确保密钥的新鲜性(Key Freshness)，这是 TA 的义务。

Kerberos 协议是一个基于对称密钥的流行的密钥服务系统。在此系统中，每个用户 U 和可信任中心(对称密钥分发中心)共享一个 DES 密钥。密钥交换协议如下：

(1) 用户 U 为了与用户 V 通信，先向 TA 申请一个会话密钥。

(2) 可信任中心 TA 随机地选取一个会话密钥 K，一个时戳 T 和生存期 L。

(3) 可信任中心用自己与 U 的密钥 k_U 加密 ID(V)等信息：

$$m_1 = E_U(K, \mathrm{ID(V)}, T, L) \tag{6-1}$$

再用自己与 V 的密钥 k_V 加密 ID(U)等信息：

$$m_2 = E_V(K, \mathrm{ID(U)}, T, L) \tag{6-2}$$

最后将 m_1 和 m_2 发送给 U。这里，ID(U)是关于 U 的某些识别信息。

(4) 用户 U 首先解密 m_1 以获得 K、ID(U)、T 和 L，接着 U 就可以通过 T 和 L 来验证密钥 K 是否在有效期内，然后用会话密钥 K 加密：

$$m_3 = E_K(\mathrm{ID(U)}, T) \tag{6-3}$$

将 m_3 和 m_2 一起发给 V。

(5) 用户 V 首先解密 m_2 以获得 ID(U)、K、T 和 L，然后用 K 解密 m_3 获得 ID(U)和 T。通过判断两个 ID(U)是否一致，从而验证了会话密钥的正确性。此后用 K 计算：

$$m_4 = E_K(T+1) \tag{6-4}$$

并发给 U。

(6) 用户 U 用 K 解密 m_4，以验证 $T+1$ 的正确性，表明 V 已收到 K。

以上协议中，m_1 和 m_2 用来提供会话密钥在传输中的秘密性；m_3 和 m_4 用来使 U 和 V 确信收到了同一把会话密钥；而 T 和 L 用来防止主动攻击者用存储的旧消息进行重复攻击。

Kerberos 协议的缺点之一是系统必须统一时钟，另一个缺点是仍不能解决对称密钥普

遍存在的 n^2 问题。

2. 非对称系统的密钥分配

1) *Blom* 方案

① 公开一个素数 p，每个用户公开一个元素 $r_U \in Z_p$，各用户的 r_U 必须互不相同。

② 可信任中心随机选择三个随机元素 a，b，$c \in Z_p$（未必不同），构成多项式：

$$f(x, y) = (a + b(x + y) + cxy) \bmod p \tag{6-5}$$

显然，它对于 x 和 y 对称：

$$f(x, y) = f(y, x) \tag{6-6}$$

③ 对用户 U，可信任中心计算：

$$a_U = a + br_U, \quad b_U = b + cr_U \tag{6-7}$$

由此构造函数：

$$g_U(x) = f(x, r_U) = (a + bx + br_U + cr_U x) \bmod p = (a_U + b_U x) \bmod p \tag{6-8}$$

并用安全信道将此线性函数传给用户 U。同样，每个用户都从可信任中心获得自己的 $g_i(x)$。

④ 如果 U 想与 V 通信，U 计算 $K_{UV} = g_U(r_V)$，而 V 计算 $K_{VU} = g_V(r_U)$，不难得知：

$$\begin{aligned} K_{UV} = K_{VU} &= f(r_U, r_V) = (a + br_U) + (b + cr_U)r_V \bmod p \\ &= a + b(r_U + r_V) + cr_U r_V \bmod p \end{aligned} \tag{6-9}$$

于是 U 和 V 便有共同的密钥，是各自通过计算得到的。

问题是其他用户，比如 W 能得知 K_{UV} 吗？假设 W 已经有了来自可信任中心的函数：

$$g_W(x) = (a_W + b_W x) \bmod p \tag{6-10}$$

他想要推知 K_{UV}。

现在，r_U 和 r_V 是公开的，而 a、b、c 是保密的。显然，W 由方程组：

$$\begin{cases} a + r_W b = a_W \\ b + r_W c = b_W \end{cases} \tag{6-11}$$

是无法确定三个变量 a、b、c 的。他得不到 a、b、c 也就求不出 K_{UV}。

然而如果任何两个用户合作，却可以唯一地确定 a、b、c。因为由联立方程：

$$\begin{cases} a + r_W b = a_W \\ b + r_W c = b_W \\ a + r_X b = a_X \\ b + r_X c = b_X \end{cases} \tag{6-12}$$

就可以确定 a、b、c，从而求出任何一个用户的密钥。

此方案的安全性如此脆弱是因为这只是规模 $k=1$ 的 Blom 方案。现将协议第②步做如下改动，信任中心即可使用如下形式的多项式：

$$\sum_{i=0}^{k} \sum_{j=0}^{k} a_{ij} x^i y^j \bmod p \tag{6-13}$$

这里 $a_{ij} \in Z_p (0 \leqslant i \leqslant k, 0 \leqslant j \leqslant k)$，并对所有的 i 和 j，有 $a_{ij} = a_{ji}$。这样便构成规模为 $k(\neq 1)$ 的 Blom 方案，此时，任何少于 k 个的用户合作，均无法破译系统密钥，只有 $k+1$ 个用户合作才可以破译密钥。

2）*Diffie-Hellman* 密钥预分配方案

① 公开一个素数 p 和一个本原元素 $\alpha \in Z_p^*$。

② ID(U)表示网络中用户 U 的某些信息，每个用户有一个秘密指数 a_U 和一个相应的公钥：

$$b_U = \alpha^{a_U} \bmod p \tag{6-14}$$

③ 可信任中心有一个签名方案 Sig_{TA}，当用户 U 入网时，可信任中心给 U 发一个签名的证书：

$$C(U) = \{ID(U), b_U, \mathrm{Sig}_{TA}(ID(U), b_U)\} \tag{6-15}$$

可信任中心无须知道 a_U 的值，证书可以放在公开的数据库中，也可以由 U 自己存储。

④ V 使用自己的私钥 a_V 和从 U 的证书中获得的公钥 b_U 计算：

$$K_{UV} = \alpha^{a_U a_V} \bmod p = b_U^{a_V} \bmod p \tag{6-16}$$

⑤ U 使用自己的私钥 a_U 及从 V 的证书中获得的公钥 b_V 计算：

$$K_{UV} = \alpha^{a_U a_V} \bmod p = b_V^{a_U} \bmod p \tag{6-17}$$

于是，通过不对称密钥体系，使 U 和 V 得到了相同的密钥。算法的安全性是基于离散对数的复杂性的。

6.2　密钥共享(密钥分配问题)[2]

6.2.1　问题的提出

有一项绝密文件锁在保险柜里，为安全起见，规定课题组 6 人中至少 4 人在场方可打开柜子，同时也可避免万一某一位丢失钥匙而造成严重事故。究竟应当怎样配备锁子和钥匙呢？不妨这样考虑锁子的数量：6 人中 3 人到场的情况是 $C_6^3 = 20$ 种，这时至少还有一把锁不能打开，最不利的情况是这 20 种组合中的不能打开的锁是各不相同的，所以至少应有 20 把锁。然后考虑钥匙的数量：先任意指定 1 人必须在场，6 人当中的 4 人在场等价于非指定的 5 人中 3 人在场，这样的情况是 $C_5^3 = 10$ 种，这 10 种组合中每种都尚有一把锁必须等指定的那个人到场才能打开。而最不利的情况是 10 种组合中的最后一把锁是各不相同的，因此指定的人至少应当有 10 把不同的钥匙去一一配合这 10 把不同的锁。无论先指定谁都一样。由此可见，每个人至少应持有 10 把不同的钥匙。6 个人共应有 60 把钥匙，平均每把锁配有 3 把钥匙。这 60 把钥匙在 6 个人中被恰当地分配。

把这个问题用到密钥分享方面，就相当于取一个 20 位的数字串，分给 6 个人保管，每人保管其中不相同的 10 位。任意 3 人组合，一定有一位并且只有一位数字是 3 个人都不掌握的，所以 3 个人不能得到完整的密钥。第四个人如果参与，虽然他与原来 3 个人中的任何两人可构成三种 3 人组合，但各组合都仍然各缺一个数位且只缺一位。设缺的分别是第 i、j、k 位，而原来的 3 人组合所空缺的位是第 l 位，因为不同 3 人组合的缺位是不同的，i、j、k 都不是 l，所以 l 一定掌握在第四个人手里，4 人组合后，l 位必然不空缺，这就保证了 4 人组合必然得到完整的密钥。

例如，密钥是 $(a_1 a_2 a_3 \cdots a_{19} a_{20})$，且：

A_1 掌握($a_{11}a_{12}a_{13}a_{14}a_{15}a_{16}a_{17}a_{18}a_{19}a_{20}$)；$A_2$ 掌握($a_5a_6a_7a_8a_9a_{10}a_{17}a_{18}a_{19}a_{20}$)

A_3 掌握($a_2a_3a_4a_8a_9a_{10}a_{14}a_{15}a_{16}a_{20}$)；$A_4$ 掌握($a_1a_3a_4a_6a_7a_{10}a_{12}a_{13}a_{16}a_{19}$)

A_5 掌握($a_1a_2a_4a_5a_7a_8a_{11}a_{13}a_{15}a_{18}$)；$A_6$ 掌握($a_1a_2a_3a_5a_6a_8a_{11}a_{12}a_{14}a_{17}$)

则第一种 3 人组合 $A_1A_2A_3$ 缺少 a_1 位，第二种 3 人组合 $A_1A_2A_4$ 缺少 a_2 位，第 i 种 3 人组合缺少 a_i 位，直到第二十种 3 人组合 $A_4A_5A_6$ 缺少 a_{20} 位。共有 $C_6^3=20$ 种不同的 3 人组合，每种都缺一个不同的位。

普遍地说，如果系统的安全性最终取决于一个主密钥，若这个主密钥偶然或蓄意地被暴露，整个系统就会受到致命攻击；若主密钥丢失或损坏，则系统中所有信息也就用不成了。在这种情况下就要使用密钥共享(Secret Key Share Scheme)方案：主密钥 k 分成了 n 个子密钥 $k_1k_2\cdots k_n$，由 n 个人分别保管，要求：

(1) 已知其中任意 t 个子密钥($t<n$)，容易算出 k。

(2) 已知任意 $t-1$ 或更少的子密钥，则不能算出 k。这种方案叫做(t，n)门限(Threshold)密钥共享方案。

6.2.2 Shamir 门限方案

1979 年，Shamir 基于拉格朗日内插多项式算法，提出了一个(t，n)门限方案。

设 GF(q)是一有限域，共享密钥 $k\in$GF(q)。可信任中心给 n($n<q$)个共享者 A_i($1\leqslant i\leqslant n$)分配共享密钥的过程如下：

(1) 可信任中心随机选择一个 $t-1$ 次多项式：

$$h(x)=a_{t-1}x^{t-1}+a_{t-2}x^{t-2}+\cdots+a_2x^2+a_1x+a_0 \tag{6-18}$$

其中，$h(x)\in$GF(q)[x]，$a_0=k$ 就是主密钥。

(2) 可信任中心在 GF(q)中选择 n 个非零的、互不相同的元素 x_1，x_2，…，x_n，分别计算：

$$y_i=h(x_i)\qquad i=1,2,\cdots,n \tag{6-19}$$

也就是找出曲线 $h(x)$上的 n 个点。

(3) 把第 i 个点，即子密钥(x_i，y_i)分配给第 i 个共享者 A_i，其中 x_i 是公开的，y_i 是属于 A_i 的私有子密钥，i=1，2，…，n。

由于 $h(x)$是 $t-1$ 次曲线，因此如果知道了 t 个点(x_s，y_s)($1\leqslant s\leqslant t$)，曲线 $h(x)$就可以按拉格朗日插值公式表达出来：

$$h(x)=\sum_{s=1}^{t}y_s\prod_{\substack{j=1\\j\neq s}}^{t}\frac{x-x_j}{x_s-x_j} \tag{6-20}$$

而 $h(x)$在 $x=0$ 点的值就是密钥：

$$k=h(0)=\sum_{s=1}^{t}y_s\prod_{\substack{j=1\\j\neq s}}^{t}\frac{-x_j}{x_s-x_j}=\sum_{s=1}^{t}b_sy_s \tag{6-21}$$

式中：

$$b_s=\prod_{\substack{j=1\\j\neq s}}^{t}\frac{-x_j}{x_s-x_j} \tag{6-22}$$

由于各 x_i 的值是公开的，因此 b_s 可以预先算出来，以加快运算速度。

【例 1】 由拉格朗日多项式 $h(x)=(7x^2+2x+11) \bmod 19$ 设计一个$(3, 5)$门限方案。若选择 $x_1=1$，$x_2=2$，$x_3=3$，$x_4=4$，$x_5=5$，试为这 5 个用户分配密钥，并分析重构密钥 k 的过程。

解： 根据已知的 x_i，可算出：

$$y_1=h(1)=1,\ y_2=h(2)=5,\ y_3=h(3)=4,\ y_4=h(4)=17,\ y_5=h(5)=6$$

这样就可以把这 5 对子密钥分配给 5 个用户保管。当其中任意 3 个用户到场，比如已知 3 个子密钥(x_2, y_2)、(x_3, y_3)和(x_5, y_5)时，则可以重构 $h(x)$。式$(6-20)$的三项分别为

$$5\frac{(x-3)(x-5)}{(2-3)(2-5)}=5\frac{(x-3)(x-5)}{(-1)(-3)}=5(x-3)(x-5)\cdot(3^{-1} \bmod 19)$$
$$=5(x-3)(x-5)\cdot 13=65(x-3)(x-5)$$

$$4\frac{(x-2)(x-5)}{(3-2)(3-5)}=4\frac{(x-2)(x-5)}{(1)(-2)}=4(x-2)(x-5)\cdot((-2)^{-1} \bmod 19)$$
$$=4(x-2)(x-5)\cdot 9=36(x-2)(x-5)$$

$$6\frac{(x-2)(x-3)}{(5-2)(5-3)}=6\frac{(x-2)(x-3)}{3\times 2}=6(x-2)(x-3)\cdot(6^{-1} \bmod 19)$$
$$=6(x-2)(x-3)\cdot 16=96(x-2)(x-3)$$

所以

$$h(x)=[65(x-2)(x-3)+36(x-2)(x-5)+96(x-2)(x-3)] \bmod 19$$
$$=(26x^2-188x+296) \bmod 19=7x^2+2x+11$$

所以 $k=11$。

6.2.3　Asmuth-Bloom 门限方案

1980 年，Asmush 和 Bloom 基于中国剩余定理提出了一个门限方案。在这个方案中，欲共享的是与密钥相联系的一个同余类。

令 p，d_1，d_2，…，d_n 是满足下列条件的一组正整数：

(1) $p>k$(k 是欲共享的密钥)。

(2) $d_1<d_2<\cdots<d_n$。

(3) 对所有的 i，有 $\gcd(p, d_i)=1$；对所有的 $i\neq j$，有 $\gcd(d_i, d_j)=1$。

(4) $d_1d_2\cdots d_t>pd_nd_{n-1}\cdots d_{n-t+2}$，于是，若 $M=d_1d_2\cdots d_t$，则$\frac{M}{p}$大于任意 $t-1$ 个 d_i 之积。(因它大于最大的 $t-1$ 个 d_i 之积。)

令 γ 是区间$\left[0, \left[\frac{M}{p}\right]-1\right]$中的一个随机整数，$\left[\frac{M}{p}\right]$表示不大于$\frac{M}{p}$的最大整数。公布 p 和 γ，计算 $k'=k+\gamma p$，则 $k'\in[0, M-1]$。将 k'分为 n 个共享密钥：$k_i=k' \bmod d_i(i=1, 2\cdots n)$。将 k_i 分配给用户 i。为了恢复 k'，只需求解中国剩余定理的联立方程组即可

$$\begin{cases} k'=k_{i_1} \bmod d_{i_1} \\ k'=k_{i_2} \bmod d_{i_2} \\ \quad\vdots \\ k'=k_{i_t} \bmod d_{i_t} \end{cases} \tag{6-23}$$

t 个方程联立，便可得到唯一解 k'，于是密钥 $k=k'-\gamma p$，即 $k=k' \bmod p$。若只有 $t-1$ 个方程，则有无数多个解。有了(4)，就制约了 n 与 t 之间并不独立。

【例 2】 密钥 $k=5$，要分配给 $n=3$ 个终端，要求若少于 $t=2$ 个终端就无法得到 k。设计一个 Asmuth-Bloom 方案。

解：取常数 $p=7(>k)$，并找 3 个与 p 互素且自相互素的数，此处找 $d_1=11$，$d_2=12$，$d_3=17$，且满足：

$$d_1 d_2 = 11 \times 12 = 132 > p d_3 = 7 \times 17 = 119$$

任意选 $\gamma=14$，满足：

$$\gamma < \left[\frac{M}{p}\right] = \left[\frac{132}{7}\right] = 18$$

于是有

$$k' = k + \gamma p = 5 + 14 \times 7 = 103$$

在[0，131]区间内求出子密钥，分配给第一个终端：

$$k_1 = 103 \bmod 11 = 4$$

分配给第二个终端和第三个终端的子密钥是：

$$k_2 = 103 \bmod 12 = 7$$

$$k_3 = 103 \bmod 17 = 1$$

现在，若有两个终端 $k_1=4$，$k_2=7$ 在场，则 k' 满足同余方程：

$$\begin{cases} k' = 4 \bmod 11 & (\text{因为 } d_1 = 11) \\ k' = 7 \bmod 12 & (\text{因为 } d_2 = 12) \end{cases}$$

解同余方程的办法是先求 $M=d_1 d_2=11\times12=132$，由此：

$$M_1 = \frac{M}{d_1} = \frac{132}{11} = 12,\ M_2 = \frac{M}{d_2} = \frac{132}{12} = 11$$

则由

$$y_1 M_1 = 1 \bmod 11$$

可求出 $y_1=1$；由 $y_2 M_2=1 \bmod 12$ 可求出 $y_2=11$。所以

$$k' = k_1 M_1 y_1 + k_2 M_2 y_2 = 4 \times 12 \times 1 + 7 \times 11 \times 11 = 895 \bmod 132 = 103$$

最后

$$k=k'-\gamma p=103-14\times7=103-98=5$$

【例 3】 数据库可简单地看做是记录的集合，记录由字段构成。若对记录加密，而个别字段又可独立解密而不涉及其他字段的安全性，则可采取共享密钥方案。

(1) 请为此设计一个 Asmuth-Bloom 密钥方案。

(2) 如一条记录 R 有 4 字段：$f_1=5$，$f_2=7$，$f_3=8$，$f_4=12$，试根据(1)中方案分析加密、解密过程。

解：(1) 设记录有 n 个字段 $f_1 f_2 \cdots f_n$，各个字段都是 0，1 符号串，可看做是一个二进制数。选 n 个不同的素数 $p_1 p_2 \cdots p_n$ 使 $p_i > f_i (i=1, 2, \cdots, n)$，整个记录的密文满足同余方程组：

$$C = f_i \bmod p_i \qquad i = 1, 2, \cdots, n \tag{6-24}$$

令

$$M = p_1 p_2 \cdots p_n,\ M_i = \frac{M}{p_i} \qquad i = 1, 2, \cdots, n \tag{6-25}$$

若
$$y_i M_i = 1 \bmod p_i \qquad i = 1, 2, \cdots, n \tag{6-26}$$
则令
$$e_i = M_i y_i \qquad i = 1, 2, \cdots, n \tag{6-27}$$
为 n 个共享子密钥。当 n 个 e_i 都齐全时，可求出整条记录的密文为
$$C = \sum_{i=1}^{\infty} e_i f_i \bmod M \qquad 0 \leqslant C < M \tag{6-28}$$
若想只检索 C 中第 i 个字段，则可由 $C = f_i \bmod p_i$ 求出 f_i。

（2）如一条记录 R 有 4 个字段，分别为 $f_1 = 5$，$f_2 = 7$，$f_3 = 8$，$f_4 = 12$，选 4 个素数，分别为 $p_1 = 7$，$p_2 = 11$，$p_3 = 13$，$p_4 = 17$，满足 $p_i > f_i$。若 C 为密文，则同余方程是：
$$\begin{cases} C = 5 \bmod 7 \\ C = 7 \bmod 11 \\ C = 8 \bmod 13 \\ C = 12 \bmod 17 \end{cases}$$
根据中国剩余定理，有
$$M = 7 \times 11 \times 13 \times 17 = 17\ 017$$
而
$$M_1 = \frac{M}{p_1} = 2431,\ M_2 = \frac{M}{p_2} = 1547$$
$$M_3 = \frac{M}{p_3} = 1309,\ M_4 = \frac{M}{p_4} = 1001$$
根据模逆元关系：
$$y_i M_i = 1 \bmod p_i$$
不难求出：
$$y_1 = 4,\ y_2 = 8,\ y_3 = 3,\ y_4 = 8$$
于是 4 个写入子密钥为
$$e_1 = y_1 M_1 = 2431 \times 4 = 9724$$
$$e_2 = y_2 M_2 = 1547 \times 8 = 12\ 376$$
$$e_3 = y_3 M_3 = 1309 \times 3 = 3927$$
$$e_4 = y_4 M_4 = 1001 \times 8 = 8008$$
所以
$$C = \sum_{i=1}^{4} e_i f_i = 9724 \times 5 + 12\ 376 \times 7 + 3927 \times 8 + 8008 \times 12$$
$$= 262\ 764 \bmod 17\ 017 = 7509$$
若只对 f_2 解密，只要用读出子密钥 $p_2 = 11$ 即可：
$$f_2 = 7509 \bmod 11 = 7$$
若要将 $f_2 = 7$ 改写为 $f_2 = 8$，则只要用写入子密钥 e_2 即可：
$$\overline{C} = \sum_{j=1}^{4} e_j f_j = [C + e_2(8-7)] \bmod 17\ 017$$
$$= [7509 + 12\ 376 \times 1] \bmod 17\ 017 = 2868$$

6.3 密 码 协 议

随着网络应用普及到各个领域，人们越来越清楚地认识到协议在网络环境下的重要性。协议是双方(或多方)为了共同完成某项任务而预先设定的一系列步骤，像程序一样按顺序执行，包括当遇到不同情况时的处理方法和分支路线。协议是具体的，靠通信系统的硬件与软件来实现，不管是友善双方还是敌对双方，为了完成通信过程，都必须遵守协议。

协议是系统的灵魂，系统所有软、硬件设备都是按照协议的步骤来工作的，因此协议应当先于系统软、硬件进行设计。密码协议是设计保密体制的核心，协议首先应当逻辑严密，科学合理，必须尽量周密、完备，一旦考虑不周，出现疏漏，就会留下安全隐患，成为攻击者成功的突破口。其次，也要考虑实际可行性与方便简洁性，以节省成本。

本节从分析一些已知协议的漏洞入手，提出若干具有实用价值和理论意义的协议，供参考。

6.3.1 RSA 协议

用 RSA 公钥密码进行可认证的保密通信，协议如下：

(1) A 先用自己的私钥对信息进行加密，得 $s=D_A(m)$。

(2) A 再用 B 的公钥对 s 进行加密，得 $c=E_B(s)=E_B(D_A(m))$。

(3) A 将得到的密文 c 发给收信人 B。

(4) B 先用自己的私钥解密密文 $D_B(c)=D_B(E_B(s))=s=D_A(m)$。

(5) B 再用 A 的公钥对 s 解密，得 $E_A(s)=E_A(D_A(m))=m$。

为了使 A 了解 B 是否完整地收到了他发的消息，协议为

(1) A 发送给 B 密文 c_1，$c_1=E_B(D_A(m))$。

(2) B 收到密文后，作 $E_A(D_B(c_1))=m$ 运算。

(3) B 送回 A 密文 c_2，$c_2=E_A(D_B(m))$。

(4) A 收到 c_2 后，作 $E_B(D_A(c_2))=\overline{m}$ 运算。

(5) A 比较 $\overline{m}$ 和 m，若二者相同，则表明 B 确已完整地收到 m。

其实这个协议是不安全的，比如 F 截获了 A 送给 B 的密文 $c_1=E_B(D_A(m))$后，将 c_1 送给 B，并称是 F 发给 B 的，如果 B 盲目地按协议执行第(2)步，就得到

$$E_F(D_B(c_1)) = E_F(D_B(E_B(D_A(m)))) = E_F(D_A(m)) = x$$

x 实际是乱码，但是如果 B 不假思索地按习惯再执行第(3)步，就得到

$$E_F(D_B(x)) = E_F D_B(E_F D_A(m)) = y$$

并将它送回 F，则 F 只用自己的私钥和 A、B 的公钥就可得到 A 发给 B 的明文：

$$E_A D_F E_B D_F(y) = E_A D_F E_B D_F(E_F D_B E_F D_A(m)) = m$$

因此，B 收到密文后，一定不要盲目地进行“回执”，而是要观察解密的结果是否有意义，阅读明文后再决定是否“回执”，这样就可避免 F 的攻击。

6.3.2 沙米尔(Shamir)协议

用 Shamir 协议进行保密通信的协议如下：

(1) A 先用自己的加密密钥对明文进行加密，得 $E_A(m)$，并将它发送给 B。

(2) B 用自己的加密密钥再加一次密，得 $E_B(E_A(m))$，并发回给 A。

(3) 如果二次加密是可以交换的，即 $E_B(E_A(m))=E_A(E_B(m))$，则 A 可将自己的加密解开，即 $D_A(E_A(E_B(m)))=E_B(m)$，并将结果发回给 B。

(4) B 用自己的解密密钥解出明文，即 $D_B(E_B(m))=m$。

在此协议中，双方都不需要公开或交换任何密钥，即使加、解密使用的是同一把对称密钥也行。就如同 A 把东西放进箱子后上了锁，B 再上另一把锁，然后 A 打开自己的锁，B 再打开自己的锁。显然此系统只能用于保密，不能进行认证。而且要求二次加密的可交换性，这并非总能成立的。

指数运算方案是可交换的，因而也可用于 Shamir 协议：

$$E_BE_A(m) = ((m)^{e_A})^{e_B} = ((m)^{e_B})^{e_A} = E_A(E_B(m))$$

取 p 为大素数，$\varphi(p)=p-1$，各用户选与 p 互素的 e，并计算其逆元 $ed=1 \bmod \varphi(p)$，(e, d)均由用户自己保管，这样就可用 Shamir 协议进行通信了。

若一次一密体系是可交换的，则用于沙米尔协议上会得到不安全的结果。

设 A 的密钥是：

$$k_A = k_1k_2\cdots k_n\cdots$$

B 的密钥是：

$$k_B = k_1'k_2'\cdots k_n'\cdots \quad (\text{各位都是 0 或 1})$$

明文是：
$$m=m_1m_2\cdots m_n\cdots$$

于是密文为
$$c_1=c_1^{(1)}c_2^{(1)}\cdots c_n^{(1)}\cdots$$

式中：
$$c_i^{(1)}=m_i\oplus k_i \qquad i=1, 2, \cdots, n, \cdots$$

A 将 c_1 发给 B，B 做如下运算：

$$c_i^{(2)} = c_i^{(1)} \oplus k_i' = m_i \oplus k_i \oplus k_i' \qquad i = 1, 2, \cdots, n, \cdots$$

B 将 c_2 发回 A，A 将 c_2 与 k 按位模 2 加：

$$c_i^{(3)} = c_i^{(2)} \oplus k_i = m_i \oplus k_i \oplus k_i' \oplus k_i = m_i \oplus k_i' \qquad i = 1, 2, \cdots, n, \cdots$$

攻击者可将 c_1、c_2、c_3 作按位模 2 加，即可得到 m：

$$c_i^{(1)} \oplus c_i^{(2)} \oplus c_i^{(3)} = (m_i \oplus k_i) \oplus (m_i \oplus k_i' \oplus k_i) + (m_i \oplus k_i \oplus k_i' \oplus k_i)$$

因自己与自己的模 2 加必为 0，所以上式的结果为 m 。这表明简单的模 2 加用于 Shamir 协议是不安全的。

6.3.3　不可否认签名协议

不可否认签名在验证时要求发信人必须参与，A 不参加验证时，B 便不能向第三者证明签名的正确性。在第 5 章中，我们已经给出了一个不可否认签名的协议，现在再展示一个与之算法不同的不可否认签名协议。

前提：p 为大素数，g 是本原元素，p、g 公开，A 的秘密私钥为 x。公开密钥为 $y=g^x \bmod p$，明文为 m，签名为 $z=m^x \bmod p$。

B 收到 A 发送来的明文 m 和签名 z 后，验证 z 是 A 对 m 的签名的步骤为

(1) B 选择两个小于 p 的随机数 a 和 b，计算：

$$c = m^a g^b \bmod p \tag{6-29}$$

并发给 A。

(2) A 选择小于 p 的随机数 q，计算：

$$s_1 = cg^q \bmod p \quad \text{和} \quad s_2 = (cg^q)^x \bmod p \tag{6-30}$$

并将 s_1、s_2 发送回 B。

(3) B 得到 s_1、s_2 后，将 a、b 送给 A 以证明步骤(1)不曾欺骗 A。

(4) A 将 q 送给 B，B 验证：

$$s_1 = cg^q \bmod p \quad \text{和} \quad s_2 = y^{b+q} z^a \bmod p \tag{6-31}$$

是否成立，若成立则签名有效，否则无效。实际上有：

$$s_2 = y^{b+q} z^a = (g^x)^{b+q} (m^x)^a = (g^b m^a)^x (g^q)^x = (cg^q)^x \bmod p$$

6.3.4 电子投票协议

设计电子投票系统时应考虑到：

① 只有合法的投票人才可以投票(身份认证)。

② 投的票必须保密。

③ 每人只投一次。

④ 每张有效票应确保被统计在结果中。

每一投票人送交秘密身份数据及选票后，应有所显示，让投票人放心，同时又不能暴露这是他的选择。协议假定有两个机构 L 和 C，L 负责审查身份，C 负责统计结果。L 送给 C 所有投票人的身份数，此外别无联系。协议如下：

(1) A 发给 L 一条“我是 A”的信息。

(2) L 进行审查，若 A 是合法投票人，则 L 送给 A 一个身份数 $I(A)$，并将 A 从投票人集合中删去，之后 L 送给 C 身份数表格 N，它只包含 A 的身份数 $I(A)$，不含 A 的真实信息。若 A 不是合法的投票人(原集合中没有)，则给 A 送出“拒绝”信息。

(3) A 自己选择一个秘密身份数 $s(A)$，将三元组$\{I(A), s(A), v(A)\}$送给 C，其中 $v(A)$是 A 投的票，而 $s(A)$用于将来公布结果时代替 $I(A)$，为的是不暴露 A。

(4) C 从身份表 N 中找是否有 $I(A)$，若有，则从 N 中取出 $I(A)$加入到投 $v(A)$票的集合中去。若 N 中无 $I(A)$，则什么也不做。

(5) C 计算并通过网络公布结果，包括公布每个人的秘密身份数 $s(A)$和他所投的票。

本协议能实现电子投票协议应具备的①～④条要求，但不能抵御 L 和 C 联合作弊。

6.4 零知识证明

近代密钥学中一个十分引人入胜的问题是零知识证明。上节其实也已经提到。如果 P 掌握了一些机密信息，它需要向 V 证明他确实掌握了这个机密，但证明过程又不能暴露任何信息，该怎么办？比如 P 能分解一个大数 n，他找到了因子 p 和 q，如果他直接向 V 公开 p 和 q，固然可以证明他所言的真实性，然而 V 也就知道了 $n=pq$ 的秘密。这就不是零知识证明。

对此问题，可以设计如下的零知识证明协议：

（1）P 请 V 随机选择一个大数 x，计算出 $x^4 \bmod n$，将结果告诉 P。

（2）因为 P 掌握 $n=pq$，就能够算出$(x^2)^2 \bmod n$ 的结果 x^2，将结果告诉 V。

（3）V 知道求 $x^2=\sqrt{x^4} \bmod n$ 是二次同余问题，其难度等价于分解 n，P 若不掌握 p 和 q 是写不出 x^2 的。于是 V 相信 P 确实掌握 p 和 q。

在这个过程中，P 只是告诉 V x^2 是多少，这对于分解 n 一点用途也没有，因为 V 原来就知道 x。

除了大数分解的零知识证明外，存在许多零知识证明问题，如迷宫路线问题、三色地图问题、子群成员问题等。根据问题中随机变量的性质，零知识证明可分为完全零知识证明、计算零知识证明和统计零知识证明；再根据验证者是否诚实，又可分为诚实验证者零知识证明和不诚实验证者零知识证明。

6.4.1　身份证明问题

智能卡、信用卡、计算机的口令等身份鉴别技术，都要求证明者 P 直接或间接地显示他的身份 $I(\mathrm{P})$，于是，不诚实的验证者 V 就有可能从中获取 P 的 $I(\mathrm{P})$信息，进而复制或冒充 P 从事非法勾当。因此 P 需要以零知识方式证明自己的身份，使 V 相信它是 P，同时又不暴露 1 比特信息。

设 $I(\mathrm{P})$包含有关 P 的身份的 k 个保密数$c_1c_2\cdots c_k(1<c_i<p)$，为了使 V 相信 P 是 $I(\mathrm{P})$的主人，但又不能告诉他 $c_1c_2\cdots c_k$，其办法是构造另外 k 个数 $d_1d_2\cdots d_k(1\leqslant d_i<p)$，它们满足：

$$d_jc_j^2 = \pm 1 \bmod n \qquad j=1,2,\cdots,k \tag{6-32}$$

将 n 和 $d_1d_2\cdots d_k$ 告知验证者 V 或加以公开。

协议首先假定存在一可信赖机构，它的职责在于公布模数 n，$n=pq$，而保密 p 和 q。这里 p 和 q 都是模 4 余 3 的大素数。零知识证明协议如下：

（1）P 选一随机数 r，计算$\pm r^2 \bmod n$，P 取其中之一告诉 V，称之为 x。

（2）V 从$\{1,2,\cdots,k\}$中选一个子集 S 告诉 P。

（3）P 计算：

$$T_c=\prod_{j\in S}c_j \quad \text{和} \quad y=rT_c \bmod n \tag{6-33}$$

并把 y 告诉 V。

（4）V 计算：

$$T_d=\prod_{j\in S}d_j \tag{6-34}$$

后，验证：

$$x=\pm y^2T_d \bmod n \tag{6-35}$$

是否成立，若等号成立，则可进行新的一轮验证或停止，否则予以拒绝。

实际上，不难推知：

$$y^2T_d=r^2T_c^2T_d=r^2\prod_{j\in S}c_j^2d_j=\pm r^2=\pm x \bmod n$$

例如，可信赖机构公开宣布 $n=2773$。P 的机密身份 $I(\mathrm{P})$包含：

$$c_1 = 1901,\ c_2 = 2114,\ c_3 = 1509,\ c_4 = 1044,\ c_5 = 2501,\ c_6 = 119$$

而

$$c_1^2 \bmod n = 582,\ c_2^2 \bmod n = 1693,\ c_3^2 \bmod n = 448$$

$$c_4^2 \bmod n = 2262,\ c_5^2 \bmod n = 2562,\ c_6^2 \bmod n = 296$$

P **公开的数据是(由** $d_j c_j^2 = \pm 1 \bmod n$**)**:

$$d_1 = 81,\ d_2 = 2678,\ d_3 = 1207,\ d_4 = 1183,\ d_5 = 2681,\ d_6 = 2595$$

P **假设选择** $r=1221$**,计算**:

$$-r^2 \equiv -1\ 490\ 841 \equiv 1033 \bmod 2773$$

令 $x=1033$**,** P **将** x **告诉** V**。** V **选择子集**$\{1, 4, 5, 6\}$**告诉** P**。** P **计算**:

$$T_c = 1901 \times 1044 \times 2501 \times 119 = 96 \bmod 2773$$

$$y = rT_c \bmod n = 1221 \times 96 \bmod 2773 = 117\ 216 \bmod 2773 = 750$$

V **收到** y **后,计算**:

$$T_d = 81 \times 1183 \times 2681 \times 2595 \bmod 2773 = 1116 \bmod 2773$$

$$y^2 T_d = 750^2 \times 1116 = 627\ 750\ 000 \bmod 2773 = 1033 \bmod 2773$$

验证得知 $y^2 T_d = 1033 = x$**,从而通过。**

6.4.2 多方远程计算问题

A **想求出能使** $g^y = x \bmod p$ **成立的** y**,苦于无力计算离散对数** $y = \log_g x$**,求助于** B**,因为** B **有足够大的计算能力。然而** A **却不想暴露** x**,为此制定协议如下**:

(1) A **选择随机数** $\gamma < p$**,计算**

$$\xi = g^{\gamma} x \bmod p \quad (\text{用 } g^{\gamma} \text{掩盖了 } x) \tag{6-36}$$

(2) A **请求** B **算出** η**,使**

$$g^{\eta} = \xi \bmod p \quad (\text{即 } \eta = \log_g \xi \bmod p) \tag{6-37}$$

(3) B **将计算结果** η **告诉** A **后,由于**

$$g^{\eta} = g^{\gamma} x = g^{\gamma} g^{y} = g^{(\gamma + y)} \bmod p \tag{6-38}$$

因此 A **很容易算出**:

$$y = (\eta - \gamma) \bmod p \tag{6-39}$$

6.5 公钥基础设施(PKI)

6.5.1 数字证书和认证中心

1. 数字证书[24, 25, 26]

公钥密码技术不仅为信息安全提供了一种强大的加密机制,而且还提供了一种识别和认证个体的方式。然而在浩瀚如海的网络世界里,大多数人并无能力生成自己的公、私密钥对。他们从何处获取密钥?互不相识的个体又从哪里找到对方的公钥?网络上得到的公钥是否可靠?彼此之间的信任怎样建立?所有这些问题表明,公钥密码技术在实用化过程

中，首先需要解决公钥的真实性和所有权问题，需要解决密钥的管理和分发问题。于是，公钥数字证书出现了，它提供了一种系统化的、可扩展的、统一的公钥分发方法。

数字证书是一个包含公钥以及公钥持有者(当然也是相应的私钥拥有者)信息的文件，它由可信任的、公正的第三方机构——认证中心(CA)签名颁发，被各类实体(持卡人/个人、商户/企业、网关/银行等)所持有，是在网上进行信息交流及商务活动的身份证明。

数字证书中最重要的信息是个体名称、个体的公钥以及 CA 的签名。通过 CA 的签名，数字证书可以把公钥与密钥所属个体(用户名和电子邮件地址等标识信息)可靠地捆绑在一起。证书中还可以包含公钥加密算法、用途、公钥有效期限等信息，为用户提供一系列的认证和鉴别机制。数字证书可用来完成信息加密、数字签名、用户身份识别三项任务。可以说，证书是网络上安全交易的用户身份证和通行证。

互不相识的人为什么能够相信对方提供的数字证书？这是基于对认证中心 CA 的数字签名的信任。假若有人对某个 CA 有怀疑，则可以查阅该 CA 的数字证书，该证书是它的上级 CA 颁发的，有它的上级 CA 的数字签名。如果还不放心，则继续追查，直到查阅最顶级 CA 的证书，叫做根证书，它以最高权威(人或机构)的信誉担保，其可信度无疑是全社会都认可的。正如你相信微软总公司，就相信它所指派的下属部门一样，对证书链中根证书的信任，就保证了对它签发的一级级证书的信任。

由于网络应用于异构环境中，因此证书的格式在所使用的范围内必须统一。从证书遵循的标准和格式来看，CA 颁发的数字证书有 X.509 公钥证书、简单 PKI (Simple Public Key Infrastructure)证书，PGP (Pretty Good Privacy)证书、属性(Attribute)证书等。在所有格式的证书中，X.509 证书应用范围最广，如 S/MIME、IPSec、SSL/TLS 等都应用了 X.509 证书。

X.509 是由国际电信联盟(ITU-T)制定的数字证书标准，它提供了一种非常通用的证书格式，是一些标准字段和扩展项的集合，包含用户及其相应公钥的信息。X.509 目前有三个版本，X.509 第 1、2 版证书所包含的主要内容如表 6.1 所示。

表 6.1　X.509 第 1、2 版证书的内容

X.509　证　书	
Version	证书版本号
SerialNumber	证书序列号：由 CA 分配给证书的唯一的数字型标识符
Signature	签名算法标识符：指定由 CA 签发证书时所使用的签名算法
Issuer	签发机构
Validity	有效期：证书开始生效及失效的日期、时间
Subject	证书用户名
SubjectPublicKeyInfo	证书持有者公开密钥信息，包含公开密钥算法和 Hash 算法
Issuer Unique Identifier	签发者唯一标识符，第 2 版中加入，可选
Subject Unique Identifier	证书持有者唯一标识符，第 2 版中加入，可选
Issuer's Signature	签发机构对证书上述内容的签名值

X.509 第 3 版证书在第 2 版的基础上增加了扩展项，使证书能够附带额外信息。任何人都可以向权威机构申请注册扩展项。如果它们应用广泛，以后可能会列入标准扩展集中。

2. 认证中心(CA)

认证中心 CA 是数字证书的颁发者，是可信任的第三方权威机构，通常是一个被称为安全域(Security Domain)的有限群体。它提供密钥管理机制、身份认证机制以及各种应用协议，它的公钥公开。

创建证书的时候，用户首先提出申请，CA 审查用户信息，特别是验证用户的公钥是否与用户的私钥配对。若检查通过，则生成证书并用 CA 的私钥签名，然后发给用户。领到证书的用户、实体或应用程序可以使用 CA 的公钥对证书进行检验，以确认证书的可靠性。因为证书只有具有 CA 的数字签名，才能保证证书的合法性和权威性(同时也就保证了持有者的公钥的真实性)，所以 CA 必须确保证书签发过程的安全，以确保签名私钥的高度机密，防止他人伪造证书。

一个典型的 CA 系统由安全服务器、注册机构(RA)、CA 服务器、目录服务器(LDAP)和数据库服务器等组成，如图 6.2 所示。

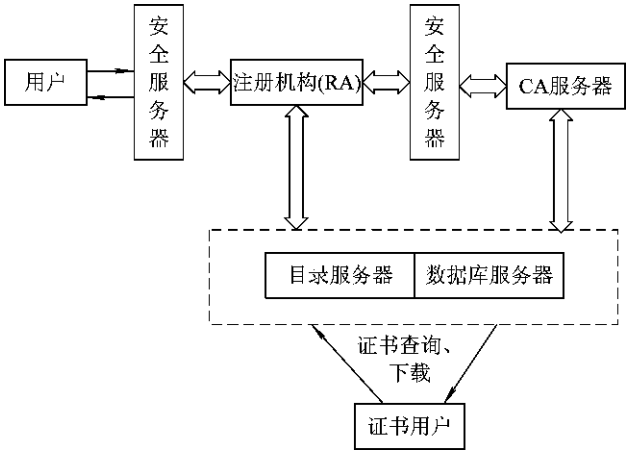

图 6.2 CA 系统的组成

· CA 服务器：CA 服务器是整个证书机构的核心，负责证书的签发。CA 首先产生自身的私钥和公钥(密钥长度至少为 1024 bit)，然后生成数字证书，并且将数字证书传输给安全服务器。

· 安全服务器：安全服务器面向普通用户，提供证书的申请、浏览，证书下载以及已撤销证书的查询等安全服务。它与用户之间的所有通信均以安全服务器的密钥加密传输。

· 注册机构(RA)：注册机构在 CA 体系结构中起承上启下的作用，一方面向 CA 转发安全服务器传输过来的证书申请请求，另一方面向目录服务器和安全服务器转发 CA 颁发的数字证书和证书撤销列表。

· 目录服务器：目录服务器提供目录浏览服务，负责将注册机构服务器传输过来的用户信息以及数字证书加入到服务器上，供用户下载和查询。

· 数据库服务器：数据库服务器用于存储和管理所有的数据(如密钥和用户信息等)、日志和统计信息。

CA 的核心功能是对证书进行签发、更新、撤销和管理，具体如下：

(1) 接收数字证书申请，验证、核查并标识申请者的身份。

（2）审批证书，决定是否受理用户数字证书的申请。

（3）签发或拒绝发放数字证书。

（4）检查证书的有效期，定期或根据用户请求对证书更新。

（5）接收用户对证书的查询、撤销。

（6）产生和发布、维护证书废止列表 CRL(Certificate Revocation Lists)。

（7）对过期数字证书、密钥等归档。

（8）其他管理功能，如 CA 的远程管理、维护，以及上级 CA 对下级 CA 的管理等。

6.5.2　PKI 概述[25] [26]

PKI(Public Key Infrastructure)是公钥基础设施的缩写。所谓基础设施，指的是为某类应用提供基本物质条件与技术支持的硬、软环境。如遍及城乡的电力基础设施使任何电器只要接通电线就能方便地获得电能；又如覆盖全球的通信基础设施使每部电话都能互通，使每一台入网的电脑都能上网。同样，基于公钥密码体制的信息安全基础设施则应能方便地为每个网络上的个体提供信息安全的技术保障与最佳服务。认证中心与公钥证书无疑是安全基础设施的两个重要组成部分，但不是 PKI 的全部。PKI 应当是一种信息安全的综合解决方案，是一套软、硬件系统和安全策略的集合，它遵循标准的密钥管理平台，结合技术和管理两方面因素，确保证书中信息的真实性，并对证书提供全程管理。

利用 PKI 系统可以使网络上的组织或实体建立安全域，并在其中生成密钥和证书。在安全域内，PKI 颁发密钥和证书，提供密钥管理(包括密钥更新、密钥恢复和密钥委托等)、证书管理(包括证书产生、更新和撤销等)和策略管理等服务。PKI 系统也允许一个组织通过证书级别或直接交换认证等方式，同其他安全域建立信任关系。通过提供一整套安全机制，使用户在不知道对方身份或分布地域很广的情况下，以证书为基础，通过一系列的信任关系进行安全通信。鉴于 PKI 为各种网络应用提供互信凭据，因此也可以说它是一个信任管理设施。

PKI 能为电子商务的应用提供如下的安全保障：

（1）消息来源认证：使之信任证书所属实体的合法身份。

（2）消息完整性认证：使之相信证书持有者所签名的文件真实无误。

（3）机密性：使之相信 PKI 采用的加密算法能够实现数据的机密传输。

（4）不可否认：使之确保发送实体和接收实体双方均不得否认自己的行为。

通过 PKI 的支持，公钥密码技术被广泛地应用于许多领域。如：

（1）S/MIME 采用 PKI 标准实现了安全电子邮件，使用了数字签名技术并支持消息和附件的加密，无需收发双发共享相同密钥。

（2）安全套接层协议 SSL 和传输层安全协议 TLS 是 Internet 网络中访问 Web 服务器的重要安全协议，它们依赖 PKI 为通信的客户机和服务器发放证书，并认证双方(或者一方)的身份，在此基础上还可以基于服务器的公钥协商会话密钥，保护后续的通信过程。

（3）IP 层安全协议 IPSec 利用网络层安全协议和建立在 PKI 上的加密与签名技术来保证通信的安全性，提供了加密和认证过程的密钥管理功能。同时，IPSec 还用于构建虚拟专用网 VPN。

6.5.3 PKI 的组成[27][28]

1995 年 10 月，因特网工程任务组(IETF－Internet Engineering Task Force)成立了 PKIX 工作组。该组织基于 X.509 证书，开发了适用于因特网的多个 PKI 标准，其内容由一系列 RFC 文档组成，称为 PKIX(即 X.509 认证的 PKI)规范。根据 RFC2501 和 RFC4210，PKI 组件模型如图 6.3 所示。

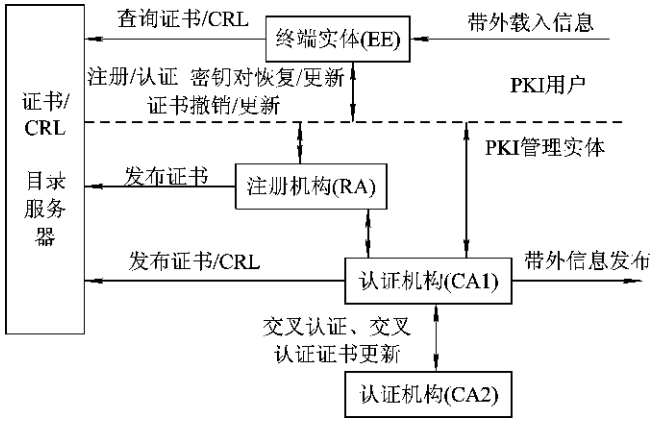

图 6.3 PKI 系统的组成

虽然基于不同的标准和应用场合，PKI 体系会有不同的结构，但是，作为一个能为互不相识的实体建立并保证信任关系的服务体系，PKI 系统应当包括认证中心、注册中心、证书库/证书撤销列表库、PKI 策略和终端实体五部分。

(1) 认证中心(CA)：负责签发证书和证书吊销列表，是证书和密钥生存周期管理过程中的核心功能部件，也是整个 PKI 系统的核心部件(虽然对于特殊的 PKI 系统，比如依据 PGP 模型建立的以用户为中心的 PKI 系统，CA 不是一个必须部分)。CA 所管辖的证书系统被称为一个“域”。终端实体在某个域注册后，当不同域中的实体需要通信时，它们通过域之间的信任关系能够建立可信任的信息交换。

(2) 注册中心(RA—Registration Authority)：RA 提供了一个用户与 CA 之间的接口。在 CA 为实体发放证书以前，RA 负责收集用户信息，对用户身份进行验证，并在 PKI 系统中为用户注册。RA 是可选的管理组件，假如没有 RA，CA 则必须承担 RA 的功能。

(3) 证书库/证书撤销列表库(Cert Repository/CRL Repository)：通常是一个目录服务器，用于存储 CA 签发的证书和证书撤销列表，并响应用户的查询请求，便于用户方便地查询证书和已吊销证书的信息。当用户的身份改变或密钥遭到破坏时，需要发布该证书已被撤销、废除的信息。证书撤销列表(CRL－Certificate Revocation List)为此提供了一种机制。CRL 是由 CA 签名的一组电子文档，包括了被吊销证书的唯一标识(即证书序列号)。

(4) PKI 策略：定义了证书管理过程和证书使用过程中的规则和约束。制定明确、合理的策略可以保证 PKI 系统正常运行。PKI 在实际应用中的性能很大程度上取决于策略的制定。一般情况下，PKI 中包括两类策略：证书策略(CP－Certificate Policy)和证书操作管理规范(CPS－Certification Practice Statement)。CP 用于管理证书的使用，在 X.509 证书

中，证书策略由唯一识别符标识，该识别符置于证书扩展项的“证书策略”域中。正是通过该项，将本证书系统所要遵循的具体操作规则与具体的策略对应起来。CPS 则是描述如何在实践中增强和支持安全策略及相应操作的过程，包括 CA 的建立和运作、证书的发放和吊销以及密钥的长度和产生过程等。

(5) 终端实体(EE－End Entity)：终端实体指数字证书持有者，也称之为用户。用户包括证书的主人和使用证书的用户。其中使用证书的用户不仅包括人类用户，还包括使用证书的某种应用，比如 IPSec 应用。严格地讲，用户不是 PKI 体系的组成部分，但它们的操作和 PKI 的功能流程密切相关。

6.5.4　PKI 中密钥和证书的管理[29～32]

证书管理和密钥管理是 PKI 的中心任务。密钥/证书从产生到废除将经历一个完整的生命周期，CA 必须具备生命周期全过程的管理功能。密钥和证书生命周期可分为初始化阶段、分发阶段和撤销阶段，各阶段又由若干过程构成，如图 6.4 所示。

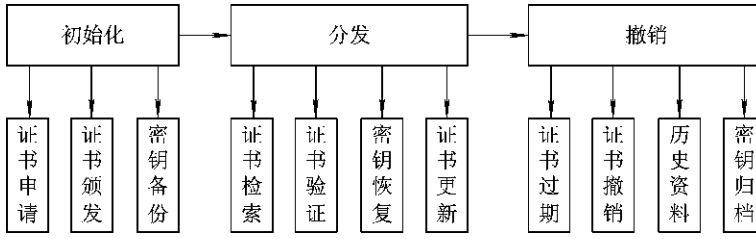

图 6.4　密钥/证书生命周期的各阶段组成

1. 证书的初始化阶段

终端实体想加入这个安全域，申请获得 PKI 服务，最先要做的事情应该是查阅 CA 的证书，甚至是查阅直至根证书的证书链，因为首先应当确认自己所申请的 CA 是合法的、可信赖的，这样才能决定是否向这个 CA 申请。一般 CA 会在自己的门户网站上发布自己的证书，用户只需登录该网站即可获取 CA 根证书。

初始化包括多个步骤，概括起来主要有三个过程，分别是申请证书、颁发证书和备份密钥。

1) 证书申请过程

终端实体任何时候都可以向 CA 发送申请证书的请求。该过程做三件事：

(1) 用户注册。终端用户首先到注册中心申请注册，注册的目的是建立公钥拥有者(实际上是与公钥相对应的私钥的拥有者)与证书所声明身份的对应关系。注册可以通过两种方式进行，一种是离线注册，即用户持相关有效证件到注册中心进行书面申请；另一种是在线注册，即用户登录 CA 提供的网站，输入身份信息进行证书申请。注册中心则应根据相应的策略审核用户申请信息。

(2) 产生密钥对。密钥对的产生有两种方式。一是由用户产生，用户通过客户端软件或使用 Web 浏览器生成。二是由 CA 产生，因为 CA 具有高度的权威性和可信性，通过 CA 生成密钥对的安全性较高。一般用户都应当有两对密钥对，一对是用于签名，另一对用于加密，每对密钥都有相应的数字证书。由于数字签名具有不可否认性，因此签名用的密钥应由 CA 产生，而加密用的密钥可以使用上述两种方式中的任一种产生。

无论哪种方式产生的密钥对，其中私钥均由用户保管，公钥和其他用户信息送交 CA 进行签名，以产生公钥证书。

（3）验证密钥对。验证密钥对的目的是确认申请者确实拥有与正在注册的公开密钥相应的私钥。用户只有向权威机构证明了自己确实拥有这个密钥对，才能获得认证机构颁发的公钥证书。PKI 把确认密钥所有权的步骤称为拥有证明（POP－Proof of Possession）。实现 POP 的方法根据证书请求中是签名密钥还是加密密钥而不同。最简单的验证方法是 CA 给用户一个随机数，要求他用私钥加密，CA 则拿用户声明的公钥解密，结果相符表明配对有效。

2）证书颁发过程

CA 把经过验证的用户公钥与用户的其他信息用自己的私钥签名，形成证书文件。证书通过物理存储介质完整地颁发到合法申请人手中，或者通过通信网络用加密方式传送给合法申请人。领到证书的用户、实体或应用程序可以使用 CA 的公钥对证书进行验证，以确认证书的可靠性和完整性。

3）密钥备份过程

为了避免密钥丢失造成严重后果，当用户证书生成时，加密密钥即被 CA 或信任第三方备份存储。但为了确保数字签名的不可否认性，不对用于签名的密钥作备份。

为防止管理人员滥用权限造成密钥泄露的危险，可以采用秘密共享方案，将备份密钥分为若干部分，由多人共同管理，任何一个管理人员都无法独自恢复该私钥。这种方法尤其适合对安全性要求特别高的情况，如 CA 根密钥的备份。对于普通用户来说，通常只对密钥加密保存即可。

2. 证书的分发阶段

一旦密钥对产生且公钥证书被颁发，密钥及证书的生命周期即已开始。PKI 提供的各项安全服务大都需要访问证书，或通过证书索取公钥，分发证书就成了 CA 频繁的日常工作。所谓证书分发，就是指交互所需证书的过程。

证书的分发包括 CA 与终端实体之间的交互以及各个不同终端实体之间的交互。CA 和终端实体之间的证书分发，主要是 CA 证书的分发，通常 CA 的证书是通过自己的门户网站来发布的，用户可以通过网络下载 CA 证书，并安装到浏览器的 CA 证书库中。

终端实体之间的证书分发通常发生于以下两种情况，一是用户之间为了进行安全通信，需要使用对方的公钥证书（证书实际上是公钥的封装）；二是当得到对方的证书后，需要验证对方的证书。这些情况下证书分发可通过物理媒介传送，或向集中存储用户证书的公共资料库查询获取，或者通过某些安全协议，如 SSL 等来交换。

证书分发阶段存在四个过程：

（1）证书检索：即通过远程资料库访问、读取证书。

（2）证书验证：在使用对方提供的证书前，应当确认该证书的完整性和有效性。其中包括交易对方证书的验证，认证中心证书的验证，以及证书链上所有证书的数字签名及有效期的验证。

（3）密钥恢复：当用户不能正常访问密钥资料时，由于留有密钥备份，因此只需向 CA 或信任第三方处提出申请，托管方就会为用户自动进行恢复。但是，签名密钥是不能恢复的。

(4) 证书更新：当一个合法的密钥对即将过期时，应当自动产生新的公/私钥对。与此同时，相应的证书也应当随之更新。

实际上，当用户注册申请、撤销证书、密钥对更新时都会导致 CA 发布新证书。

3. 证书的撤销阶段

在网络环境中，证书在有效期内也可能因多种原因需要撤销，密钥/证书生命周期就此结束。此阶段包括四方面内容：

(1) 证书过期处理：证书生命周期的自然结束。

(2) 证书撤销处理：因为密钥泄露或证书持有者状态改变等其他原因，CA 宣布一个合法证书不再有效，并消除用户与公钥的绑定。

证书撤销后，PKI 系统应及时通知终端实体使之了解证书撤销情况。目前常用的证书撤销机制有两种：一种是周期性公布机制，如证书撤销列表 CRL；另一种是在线证书查询机制，如在线证书状态协议(OCSP－Online Certificate State Protocol)。PKI 的 X 标准采用 CRL 作为证书有效期的控制机制。

CRL 的发布周期由证书操作管理规范 CPS 规定，公布间隔可能是一天也可能是一周。在 CRL 中包含有下一个 CRL 的发布时间，用户应该留意该时间，从而保证所使用的 CRL 是最新的。CRL 的发布方式有多种，一般与证书分发方式一样，如 X.500 或(LDAP－Lightweight Directory Access Protocol)目录服务器，用户从该目录获取证书的同时，可以从目录中下载最新的 CRL，检验证书的序列号是否在该 CRL 中，从而验证该证书是否有效。

(3) 历史资料保存：由于密钥更新、证书过期等原因，每个用户都可能拥有多个“旧”私钥和对应的公钥证书，这些私钥和证书统称为用户密钥历史资料。因为密钥过期后，有时仍然需要用它来解密机密数据，所以有必要可靠而安全的保存它们。

(4) 密钥资料归档：密钥归档是对 CA 和其他信任方的密钥资料(包括加密和验证证书)长期储存的方式，这种服务一般由第三方提供。出于对密钥历史恢复、审计和解决可能出现的纠纷的考虑，密钥归档对于 PKI 系统是必要的。

习　题　6

1. Diffie-Hellman 密钥交换过程中，设大素数 $p=97$，$\alpha=5$ 是本原根，U 和 V 选择的秘密指数分别是 $a_U=36$ 和 $a_V=58$。求两者的公钥分别是什么？共享密钥 K_{UV} 是什么？

2. 在一个有 A、B、C 三个用户的网络中，利用 Blom 方案，设 $p=23$，并令各用户的公开 r_U 分别是 $r_A=11$，$r_B=3$，$r_C=2$，可信任中心选择的随机数为 $a=8$，$b=3$，$c=1$，试计算任意两人的会话密钥。

3. 现有一个(3，5)Shamir 门限方案，$p=17$。选 $x_i=1$，2，3，4，5 分别代入 $h(x)$，得到 5 个秘密份额：(1，8)，(2，7)，(3，10)，(4，0)，(5，11)。若已知子密钥是(x_2, y_2)，(x_3, y_3)，(x_5, y_5)，试恢复出密钥 k。

4. 在公钥密码体制中，每一个用户 U 都有自己的公钥 P_U 和私钥 S_U。如果任意两个用户 A、B 按以下方式通信：A 发给 B 消息$(E_{PB}(m), A)$；B 收到后，自动向 A 返回消息

$(E_{PA}(m), B)$，以使 A 知道 B 确实收到报文 m。问：

(1) 用户 C 怎样通过攻击手段获取报文 m？

(2) 若通信格式变为 A 发给 B 消息 $E_{PB}(E_{SA}(m), m, A)$；B 向 A 返回消息 $E_{PA}(E_{SB}(m), m, B)$，这时的安全性如何？分析 A、B 这时如何相互认证并传递消息 m。

5. 根据下列时局设计一个 Asmuth-Bloom 门限方案。设密钥 $k=4$，分配给 $n=3$ 个用户，常数 $p=7$，取 $d_1=9$，$d_2=11$，$d_3=13$。任选 $\gamma=10$，要求少于 2 个用户时无法得到密钥 k。

实践练习 6-1

实验题目： Shamir 秘密门限共享方案的设计。

实验平台： Mathematica 4.0。

实验内容： 设计一个基于格朗日内插多项式算法的(5, 8)密钥分配方案，给 8 个用户分配秘密份额。验证只有不少于 5 人时才能得到所隐藏的秘密。

实验步骤：

(1) 秘密选择大素数 p 和一个 4 次方程式

$$f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \bmod p$$

其中，a_4、a_3、a_2、a_1 秘密选取，a_0 是需要隐藏的秘密。

(2) 任意求出曲线上的 8 个点(x_1, y_1)，$(x_2, y_2)\cdots(x_8, y_8)$，各个 x_i 是各用户的公钥，而各个 y_i 是各用户的私钥。

(3) 方法 1。求解 5 阶矩阵方程(即五元联立方程组)：

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 \\ 1 & x_5 & x_5^2 & x_5^3 & x_5^4 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} \bmod p$$

就可以求出 a_4、a_3、a_2、a_1 和 a_0

(4) 方法 2。采用拉格朗日插值算法(见公式(6-20)～(6-22))。验证：如果少于 5 个用户到场，方程数少于未知数个数，就得不到唯一解；如果等于或多于 5 个用户到场，则不论用哪 5 个用户的数据，算出的结果都是一样的。

实践练习 6-2

实验题目： PKI 的基本体系结构与安全 Web 访问。

实验平台： Windows 2000。

实验内容： 安全 Web 配置(SSL 协议)和安全邮件收发(S/MIME 协议)。

实验步骤：

(1) **了解** PKI **的基本概念，**PKI **的基本体系结构，证书的格式。**

(2) **在** Web **服务器和浏览器上配置** SSL **服务。**

(3) **利用数字证书在** Outlook Express **中发送安全邮件。**

实验步骤(2)**的提示：**

SSL **要求将服务器身份验证证书安装在** Web **服务器上，应用** SSL **时，客户端使用** HTTPS **协议并指定** http://URL，**而且服务器在** TCP **端口** 443 **上侦听。**

(1) **生成证书申请文件：**

① **在** Windows**“开始”菜单中选择“运行”，键入“**mmc**”，启动管理控制台，在“文件”菜单中打开“添加/删除管理单元”对话框，将“**Internet **信息服务”管理单元添加上。**

② **选择要安装证书的** Web **站点，点右键，然后单击“属性”→“目录安全性”→“安全通信”→“服务器证书”，启动** Web **服务器证书向导。**

③ **创建一个新证书，在“名称”字段中键入证书的描述性名称，在“位长”字段中键入密钥的位长，使用当前** Web **站点名称作为默认名称。键入组织名称和组织单位，键入站点的公用名，如** www.contoso.com**。如果是内部站点，并且用户是通过计算机名称浏览的，则输入计算机的是** NetBIOS **或** DNS **名称。在“国家/地区”、“州/省”和“城市/县市”等字段中输入正确的信息。**

④ **输入证书申请的文件名。将此申请文件发送到** CA**。**CA **验证私钥签名的信息后，将在一个文件中发回证书。**

⑤ **重新启动** Web **服务器证书向导。**

(2) **提交申请：**

① **使用“记事本”打开在前面的过程中生成的证书文件，将它的整个内容复制到剪贴板。**

② **启动** Internet Explorer**，导航到** http://hostname/CertSrv**，其中** hostname **是运行** Microsoft **证书服务的计算机的名称。**

③ **单击“申请一个证书”→“下一步”。在“选择申请类型”页，单击“高级申请”→“下一步”。在“高级证书申请”页中，单击“使用** Base64 **编码的** PKCS＃10 **文件提交证书申请”→“下一步”，在“提交一个保存的申请”页中，单击“**Base64 **编码的证书申请(**PKCS ＃10 **或** ＃7)**”文本框，按住** Ctrl＋V **组合键，粘贴先前复制到剪贴板上的证书申请。**

④ **在“证书模板”组合框中，单击“**Web **服务器”→“提交”。关闭** Internet Explorer**。**

(3) **颁发证书：**

① **启动“证书颁发机构”工具，选择“挂起的申请”文件夹。选择刚才提交的证书申请。**

② **在“操作”菜单中，单击“所有任务”→“颁发”。**

③ **确认该证书显示在“颁发的证书”文件夹中，双击查看。在“详细信息”中，单击“复制到文件”，将证书保存为** Base 64 **编码的** X.509 **证书。**

(4) **在** Web **服务器上安装证书：**

① **展开服务器名称，选择要安装证书的** Web **站点。右键单击该站点，选择“属性”。**

② **单击“目录安全性”→“服务器证书”，根据** Web **服务器证书向导安装证书。**

(5) **将资源配置为要求** SSL **访问：**

① 展开服务器名称和已安装证书的 Web 站点。

② 右键单击某个虚拟目录，然后单击“属性”→“目录安全性”→“安全通信”下的“编辑”→“求安全通道（SSL）”→“确定”。

③ 至此，客户端必须使用 HTTPS 才能浏览此虚拟目录。

第 7 章　密码学在网络安全中的应用

现代密码学伴随网络的兴起而诞生，随着网络的发展而得到广泛的应用。实用化技术不同于单纯原理性讨论，实际情况是复杂的，理论上的基本方法常被变通使用，或者是多种方法结合使用。其次，实用技术不仅应追求效果上的最佳，往往还须考虑实际可行性，综合权衡经济成本等多种因素。本章将对无线移动网络、无线局域网和 Internet 安全技术中如何应用密码学原理，怎样结合实际情况创新使用加密算法及其一些实际应用做概要介绍。

7.1　无线移动网络中的密码技术

移动通信的发展大致经历了三个阶段。第一代(1G)是模拟移动通信系统，几乎没有采用安全技术。第二代(2G)是数字移动通信系统，如欧洲全球移动通信系统 GSM(Global System for Mobile Communication)，对安全性有了较大的改进。第三代(3G)是宽带移动通信系统，定义了更加完善的安全特征与鉴权服务。

7.1.1　GSM 中的认证与加密

全球移动通信系统(GSM)是目前全世界广泛使用的数字蜂窝式无线通信系统。为了防止非法用户接入系统以及区分用户的等级，手机入网时需要登记注册，合法用户首先应购买通信公司的用户识别卡(SIM)，上面有国际移动用户识别码 IMSI(不是呼叫号码)和个人用户密码号 K_i，同样的，IMSI 和 K_i 也被保存在(或发送到)通信公司鉴权中心(AUC)。

每个移动用户都有一个唯一的国际移动用户识别码 IMSI，它只在第一次入网时传送一次，或者在出错时再使用。基站与手机之间频繁联络使用的是临时移动用户身份号 TMSI，它是由网络随机生成并临时分配给用户的一个号码，在每次通话时将 TMSI 加密后传给用户。使用 TMSI 能避免用户唯一的身份码 IMSI 的泄露。同时，每次通话使用的 TMSI 都不同，就避免了由 TMSI 窃听用户通信的可能。由于 TMSI 与用户没有确定的对应关系，因此无法根据 TSMI 跟踪用户，对用户所在位置起到了保密作用。

GSM 安全算法主要有三个，即 A_3、A_5 和 A_8，分别用于鉴权、加密和生成密钥。三个算法的实现细节是非公开的，这里仅介绍用法。GSM 系统鉴权与加、解密流程如图 7.1 所示。

1. 对用户的认证和鉴权

GSM 系统对移动用户合法性的鉴权采用了询问/响应认证协议。当移动台申请位置更新时，或者当移动台发出呼叫或被呼时，立即触发鉴权。鉴权过程主要涉及 GSM 系统的 AUC、HLR(原籍位置寄存器)、MSC/VLR(移动交换中心/访问位置寄存器)以及 MS(移

动台)。鉴权开始，MSC/VLR 向用户发送一个 128 bit 的随机数 RAND，MS 使用此 RAND 以及鉴权密钥 K_i(K_i 存储在 SIM 卡中)，通过 A_3 算法计算出符号响应 SRES2，并发回给 BS(基站)。MSC/VLR 也会用同样的 RAND 和 K_i 计算出一个符号响应 SRES1，比较两者相同与否便可决定是否允许该用户接入网络。

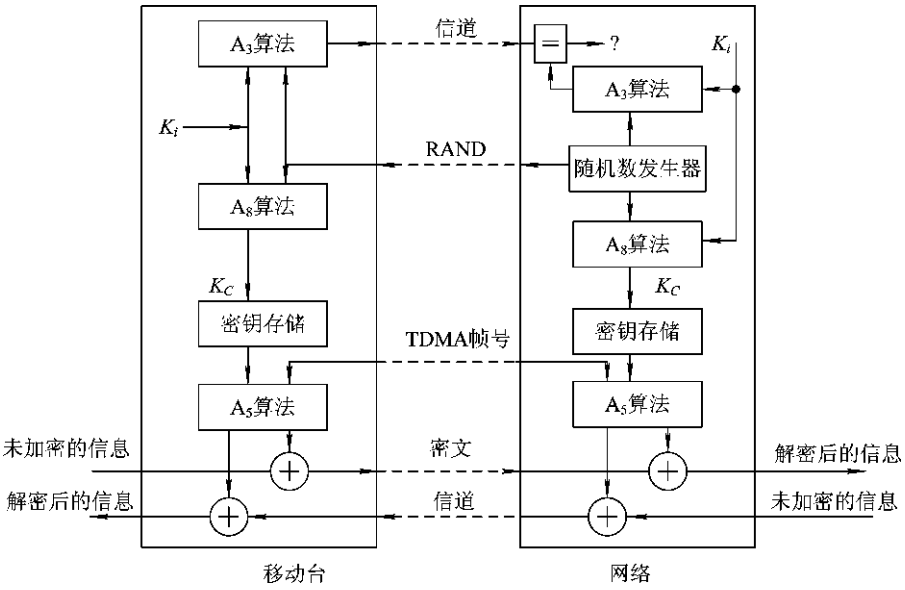

图 7.1 GSM 的加密与解密流程

2. 生成加密密钥

在鉴权程序计算 SRES 的同时，移动交换中心和移动台通过算法 A_8 也在计算加密密钥 K_C，密钥 K_C 的长度是 64 bit。A_8 的输入仍然是随机数 RAND 和鉴权密钥 K_i。由于每次使用的随机数都不同，因此每次产生的加密密钥 K_C 也不同，以增强保密性。系统启动加密模式前，移动交换中心和移动台均将 K_C 暂存。

3. 数据加密和解密

为了确保用户信息(话音或数据业务)以及与用户有关的信令信息的保密性，在 BS(基站)和 MS(手机)之间无线接入部分的信息将被加密传输。启动加密后，MSC/VLR 将把加密模式命令 M 通过 BS 发送给 MS。MS 收到后，以 M(加密模式)和 TDMA(时分多址)帧编号为输入，以 K_C 为密钥，通过加密算法 A_5 将消息加密，消息以密文方式无线发送给 BS。MSC/VLR 采用相同算法对此解密，恢复 M 以验证测试的正确性。验证通过后，无线链路上往返的信息均以 K_C 为密钥，使用 A_5 算法进行加、解密。

7.1.2 3G 中的安全算法[33~35]

国际标准化机构 3GPP(3G 合作伙伴项目，3G Partnership Project)提出的 3G 安全结构中，重点描述了网络接入的安全问题，实现了包括认证和密钥协商算法，以及与无线数据链路层信令数据、用户数据的数据加密和数据完整性算法，其核心均建立在密码技术的基础之上。安全体系中定义了 12 种密码算法，分别是：

$f0$——随机数生成函数；　　$f1$——网络鉴权函数

$f1^*$——重同步消息函数；　$f2$——用户鉴权函数

$f3$——加密密钥生成函数；　$f4$——完整性密钥生成函数

$f5$——正常情况下用的匿名密钥生成函数；$f5^*$——生成重同步时匿名密钥的函数

$f6$——加密国际移动用户识别码(IMUI)；　$f7$——对用户身份进行解密

$f8$——数据加密算法；　$f9$——数据完整性算法

密钥协商算法用于 USIM(用户业务识别模块)、VLR/SGSN(访问位置寄存器/服务 GPRS 支持节点)和 HLR(归属位置寄存器)之间的双向认证及密钥分配，利用 $f0 \sim f5^*$ 算法实现。其中：

$f0$ 算法仅在 AUC(鉴权中心)中执行，用于产生随机数 RAND。

$f1$ 算法为网络鉴权函数，用于计算网络鉴权时的 XMAC－A。

$f1^*$ 算法为重同步消息函数，支持重同步功能，保证从 $f1^*$ 的函数值无法反推出 $f1$。

$f5$ 算法为密钥生成函数，用于计算匿名密钥 AK，对鉴权序列号 SQN 加解密，以防止被位置跟踪。

$f5^*$ 算法为密钥生成函数，用于计算重同步时的匿名密钥。

$f6$ 和 $f7$ 算法用于增强用户身份保密。

$f8$ 和 $f9$ 算法对空中接口的数据进行加密和完整性保护。

上述算法中，除 $f8$ 和 $f9$ 是通过分组密码算法 KASUMI 构造得到的标准化算法外，其他算法函数还没有标准化，因此运营商可自由选择算法来满足基本要求或遵从建议中的基于 Rijndael 密码(见本书第 3 章)的解决方案。在 UE(移动用户设备)和 RNC(无线网络控制器)之间的无线链路上，$f8$ 用来对用户数据和信令数据加密；$f9$ 用来保证信令信息的完整性，并对信令来源进行认证。下面分别对 KASUMI 算法和 $f8$、$f9$ 算法做简单介绍。

1. KASUMI 算法 [36～38]

KASUMI 算法是基于日本三菱公司的分组密码 MISTY1 算法的改进版本，是一种分组加密算法，它的设计遵循了三条原则：在安全性上有足够的数学基础；算法的软件实现在任何处理器上都足够快；算法的硬件实现也足够快。这个算法在设计上能够对抗差分和线性密码分析，其安全性是可证明的，目前主要应用于 3G 的安全算法 $f8$ 和 $f9$ 之中。

KASUMI 算法中密钥长度为 128 bit，对 64 bit 的输入分组进行如图 7.2 所示的 8 轮迭代加密运算，输出长度仍为 64 bit。

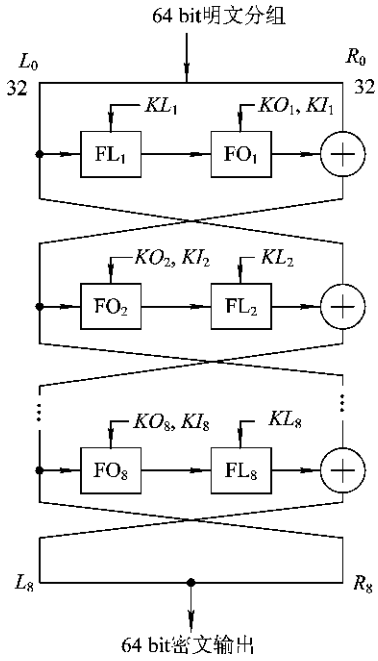

图 7.2　KASUMI 算法流程

首先，输入数据 I 被分为 32 bit 的左右两部分，即 L_0 和 R_0，第 i 轮加密算法为

$$R_i = L_{i-1},\ L_i = R_{i-1} \oplus f_i(L_{i-1},\ RK_i)$$
$$i = 1,\ 2,\ \cdots,\ 8 \qquad (7-1)$$

第 i 轮将上一轮输出的左半部 L_{i-1} 作为该轮的右半部 R_{i-1}；而第 i 轮的左半部由第 $i-1$ 轮输出的右半部 R_{i-1} 与第 i 轮的轮函数 f_i 的结果进行异或运算得到

(RK_i 为第 i 轮的子密钥)。经过 8 轮的迭代后，L_8 和 R_8 相串联得到 64 bit 的字符串输出。

其中，轮函数 f_i 以 L_{i-1} 和 RK_i 为参数，输入 32 bit，输出 32 bit。密钥 RK_i 由三个一组的子密钥(KL_i，KO_i，KI_i)组成，其中 KL_i 为 32 bit，KO_i 和 KI_i 均为 48 bit，共 128 bit。轮函数 f_i 包括一个输入、输出均为 32 bit 的非线性混合函数 FO 和一个输入、输出均为 32 bit 的线性混合函数 FL，函数 FO 以 KO_i 和 KI_i 为密钥，函数 FL 以 KL_i 为密钥。轮函数 f_i 在不同的奇偶轮有不同的表达形式。

对于 1，3，5，7 轮：

$$f_i(I, RK_i) = \mathrm{FO}(\mathrm{FL}(I, KL_i), KO_i, KI_i) \tag{7-2}$$

对于 2，4，6，8 轮：

$$f_i(I, RK_i) = \mathrm{FL}(\mathrm{FO}(I, KO_i, KI_i), KL_i) \tag{7-3}$$

下面对算法中各个函数的结构进行简单说明。

1) FL 函数

FL 函数的结构如图 7.3 所示，该函数接收并处理 32 bit 的输入数据 I 和 32 bit 的子密钥 KL_i。子密钥 KL_i 被分成两个各 16 bit 的子密钥 $KL_{i,1}$ 和 $KL_{i,2}$。同样，输入数据也被分为两个 16 bit 的左右两部分 L 和 R。定义 ROL 表示循环移位，算法可表达为

$$\begin{cases} R' = R \oplus \mathrm{ROL}(L \cap KL_{i,1}) \\ L' = L \oplus \mathrm{ROL}(R' \cup KL_{i,2}) \end{cases} \tag{7-4}$$

最后以 $R' \| L'$ 作为 32 bit 的输出值。图 7.3 中，$\cap$ 为与运算；$\cup$ 为或运算；$\oplus$ 为异或运算；$<<<$ 为循环左移 1 位的运算；$>>>$ 为循环右移 1 位的运算。$X \| Y$ 表示两序列相连接。

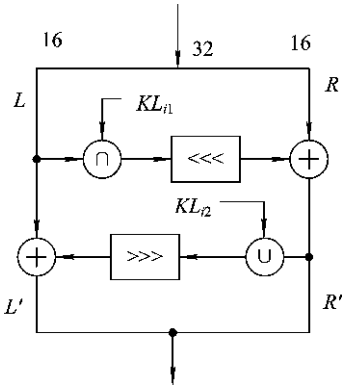

图 7.3 FL 函数

2) FO 函数

FO 函数结构见图 7.4，该函数由一个输入、输出均为 16 bit 的非线性混合函数 FI(FI 函数下段介绍)进行 3 轮重复运算而构成。FO 函数在两组密钥 KO_i 和 KI_i 控制下对 32 bit 输入进行处理。输入数据 I 被分成各 16 bit 的两部分 L_0 和 R_0，而两组 48 bit 的密钥 KO_i 和 KI_i 均被分成三个 16 bit 的子密钥 $KO_{i,1}$、$KO_{i,2}$、$KO_{i,3}$ 和 $KI_{i,1}$、$KI_{i,2}$、$KI_{i,3}$。对于三轮运算中的每一轮，算法可表达为

$$R_j = \mathrm{FI}(L_{j-1} \oplus KO_{i,j}, KI_{i,j}) \oplus R_{j-1}, \; L_j = R_{j-1} \qquad (1 \leqslant j \leqslant 3) \tag{7-5}$$

FO 函数最后得到输出是 32 bit 的 $L_3 \| R_3$。

3) FI 函数

FI 函数结构见图 7.5，它是一个由非线性处理过程构成的 4 轮结构。输入数据 I 被分为两个不等长的两部分：9 bit 的 L_0 和 7 bit 的 R_0。密钥 $KI_{i,j}$ 也被分为 7 bit 的 $KI_{i,j,1}$ 和 9 bit 的 $KI_{i,j,2}$。函数中包含了两个 S 盒。S7 为 7 bit 输入 7 bit 输出；S9 为 9 bit 输入 9 bit 输出，两者均为线性逻辑运算。处理过程中还使用了非线性扩展函数 ZE 和收缩函数 TR。ZE(x)将 7 bit 输入数据后面补两位零得到 9 bit 的输出；TR(x)将 9 bit 输入数据删除最前面两位得到 7 bit 的输出。计算过程可表示为

$$\left.\begin{aligned} L_1 &= R_0, & R_0 &= S9[L_0] \oplus \mathrm{ZE}(R_0) \\ L_2 &= R_1 \oplus KI_{i,j,2}, & R_2 &= S7[L_1] \oplus \mathrm{TR}(R_1) \oplus KI_{i,j,1} \\ L_3 &= R_2, & R_3 &= S9[L_2] \oplus \mathrm{ZE}(R_2) \\ L_4 &= S7[L_3] \oplus \mathrm{TR}(R_3), & R_4 &= R_3 \end{aligned}\right\} \quad (7-6)$$

最后以 16 bit **的** $L_4 \| R_4$ **为输出。**

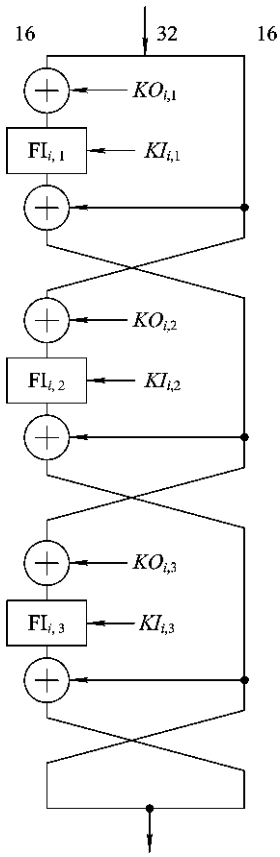

图 7.4　FO **函数**

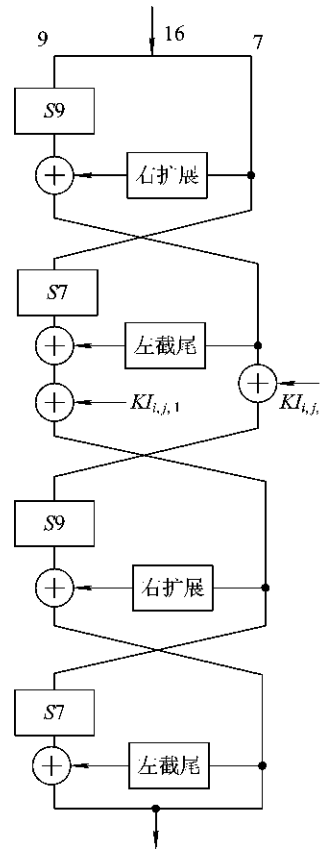

图 7.5　FI **函数**

KASUMI **算法与第** 4 **章介绍的** Camellia **算法都是由日本三菱公司参与开发的对称密钥加密算法。读者不难发现，两者的具体算法是完全不同的。**Camellia **以字节或字为操作单元，通过软件来实现，其性能更高，可以普遍适用于** 8 bit **或** 32 bit **的处理器(如智能卡或嵌入式系统等)。**KASUMI **算法则主要用于能源供给有限的移动通信系统，以确保文本或语音信息的保密性。**

2. 数据加密算法 $f8$

在移动通信中，用户数据和信令数据的保密采用序列加密方法。$f8$ **的作用实际上是一个密钥流发生器，它以密钥序列号等参数为输入，运用多级** KASUMI **算法生成序列密钥流。在发送端将待加密明文数据流与密钥流进行逐位异或运算，得到密文流；在接收端则将密文流与同样的密钥流进行异或运算，得到明文数据流。加密和解密的全过程见图** 7.6**。系统采用同步技术，使解密过程密钥流的初始位置与密文准确定位。**

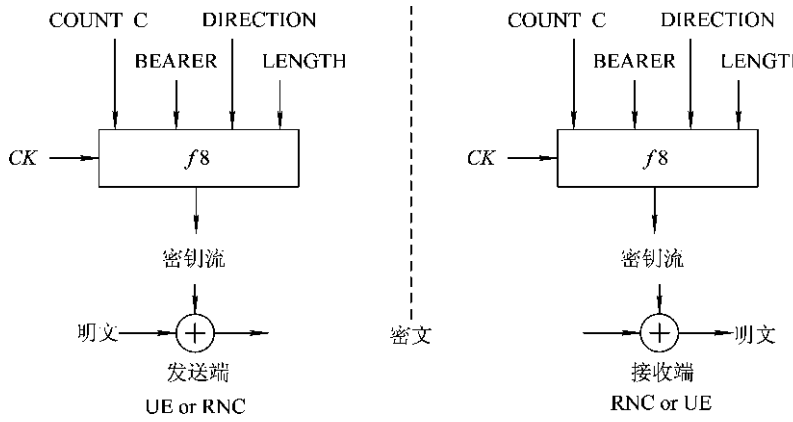

图 7.6 加、解密过程

*f*8 **算法产生密钥流的基本流程见图** 7.7。

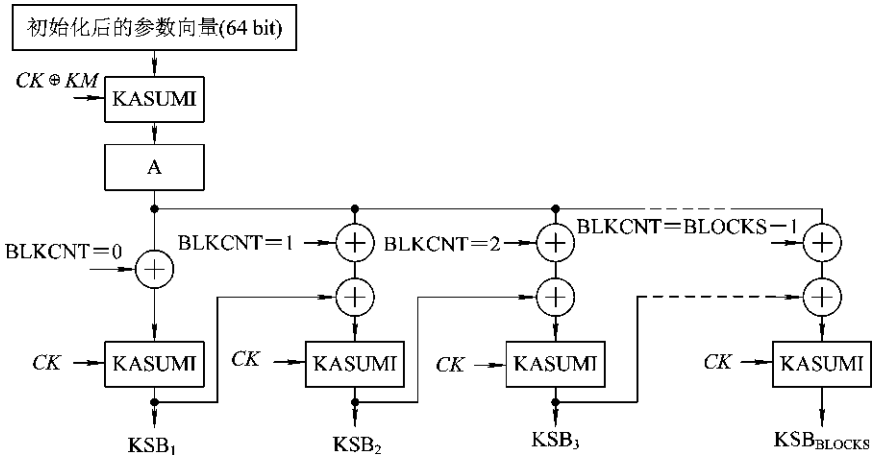

图 7.7 *f*8 算法流程

(1) **初始化：首先填写** 64 bit **的初始化参数向量，其中包括序列密钥号** COUNT－C、5 bit **的承载标识符** BEARER、1 bit **的上下行标识位** DIRECTION、**明文长度** LENGTH **等信息。设** *CK* **表示** 128 bit **的原始密钥，** *KM* **为密钥修正符(取值为十六进制数的** 32 **个** 5，**共** 128 bit)，**二者相异或后作为** KASUMI **算法模块的控制密钥，而** 64 bit **的初始化参数向量作为** KASUMI **算法模块的输入数据，计算得到** 64 bit **的输出数据，存在寄存器** A **中。**

(2) **序列密钥的分组结构：设明文长度为** LENGTH，**那么序列密钥的长度也应当有同样的长度。而** KASUMI **算法模块一次只能输出** 64 bit **的数据，因此采用分组计算，再将结果相连接。设** BLOCKS **表示分组的总数目，则** BLOCKS=[LENGTH/64]，**方括号[…]表示进位取整。用** n **表示分组序号，** $n=1, 2, \cdots,$ BLOCKS，**其中第** n **个分组记做** $\mathrm{KSB}_n[i]$，$i=0, 1, 2, \cdots, 63$。**将来这些分组连接起来之后，序列密钥** KS **与各分组的对应关系是：**

$$KS[64(n-1)+i] = \mathrm{KSB}_n[i] \tag{7-7}$$

(3) **各个分组的计算：以原始密钥** CK **为控制，通过** KASUMI **算法模块来计算产生第** n **个分组的序列密钥** $\mathrm{KSB}_n[i]$。KASUMI **算法模块的输入来自寄存器** A **的** 64 bit **数据与**

$n-1$ 的模 2 加，再加上一个分组输出的 64 bit 数据。即

$$KSB_n = KASUMI[A \oplus BLKCNT \oplus KSB_{n-1}]_{CK} \tag{7-8}$$

式中：BLKCNT$=n-1$。

3. 数据完整性算法 *f*9

为了对移动用户设备和无线网络控制中心之间无线链路上的信令信息进行完整性保护和信源认证，发送端通过 $f9$ 算法对信令信息、密钥序列号等输入参数计算出消息认证码 MAC－I，附加在信令信息后一起发送到接收端。接收端也将收到的信令数据用 $f9$ 算法进行同样的计算，算出消息认证码 XMAC－I，将它与所收到的 MAC－I 进行比较，验证数据的完整性。其过程如图 7.8 所示。

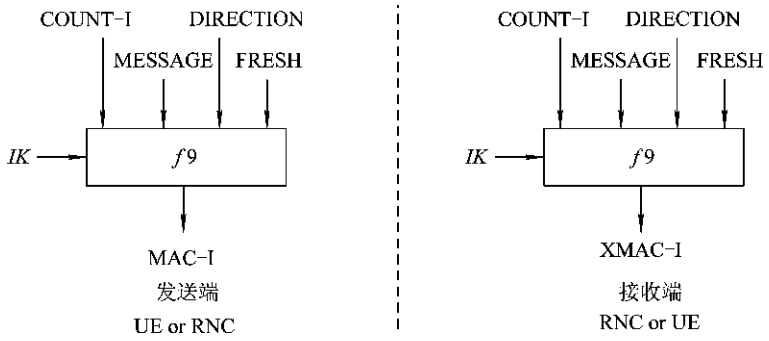

图 7.8　MAC－I/XMAC－I 生成过程

$f9$ 算法流程见图 7.9，其原理是 KASUMI 算法的反复迭代。

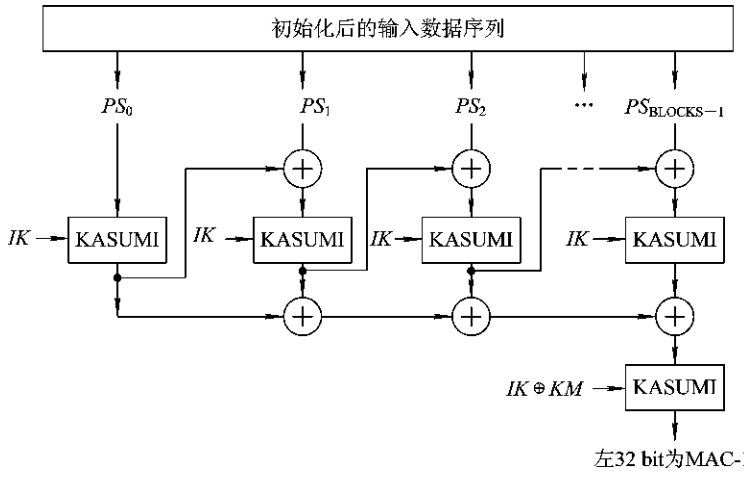

图 7.9　$f9$ 算法流程

(1) 数据分组：首先将来自无线网络控制中心(RNC)的信令数据 MESSAGE 加上前缀与后缀信息，如密钥序列号 COUNT－I、上下行标识位 DIRECTION，以及来自 RNC 的 32 bit 的随机数 FRESH，最后添上一个 1 和若干个 0，使总比特数凑够 64 的整数倍。然后按每个分组 64 bit 将数据分组。其中第 n 组表示为

$$PS_n[i] \qquad (n = 0, 1, 2, \cdots, \text{BLOCKS}-1;\ i = 0, 1, 2, \cdots, 63) \tag{7-9}$$

其中，BLOCKS 表示分组的总数目，取决于信令的长度。

(2) 迭代压缩：以 PS_0 为输入，IK 为原始密钥，通过 KASUMI 模块计算得到 64 bit 的输出数据；将它与 PS_1 模 2 加后作为第二级 KASUMI 模块的输入，仍然以 IK 为密钥，计算出第二级的输出。同样，第 n 级的输出与 PS_n 模 2 加后作为第 $n+1$ 级 KASUMI 模块的输入，计算得出第 $n+1$ 级的输出。最后将各级输出数据累加起来。

迭代工作可以在两个 64 bit 的寄存器 A 和 B 中完成：A 和 B 初始值清零，A 用来存储每级 KASUMI 模块的输出，每计算新的一级时它都被新的结果刷新：

$$A = \mathrm{KASUMI}[A \oplus PS_n]_{IK} \tag{7-10}$$

与此同时，每计算一级的时候，B 也被累加(模 2 加)一次：

$$B = B \oplus A \tag{7-11}$$

交替进行式(7-10)与(7-11)，直至处理过全部分组。

(3) 消息认证码的产生：引入密钥修正符 KM(128 bit)，赋值为十六进制数的 32 个“A”，将它与原始密钥 IK 相异或后，作为 KASUMI 模块的控制，而前面计算的结果，即 64 bit 的 B 作为 KASUMI 模块的输入，计算得到 64 bit 的输出，将右边 32 bit 舍弃，保留左边的 32 bit，这就是最终求得的消息认证码。

7.2 无线局域网中的密码技术

IEEE(电气电子工程师协会)提出一系列 WLAN(无线局域网)的技术解决方案，其中 IEEE 802.11b 是目前多数无线局域网产品支持的主流标准。为了保证移动站和 AP(接入点)之间通信的安全性，802.11b 采用了有线对等保密 WEP(Wired Equivalent Privacy)机制。这曾经是保障 WLAN 数据传输安全的核心部分。但是后来很多研究表明，WEP 协议设计本身存在一些难以克服的缺陷。为此，IEEE 802.11i 任务小组致力于开发新的安全标准，在正式版本出台前，提出了无线局域网保护性接入 WPA(Wi-Fi Protected Access)作为过渡方案。任务小组最终于 2004 年 6 月发布了 802.11i 标准。该标准增强了 WLAN 中的数据加密和认证性能，并且针对 WEP 加密机制的安全漏洞做了多方面的改进。本节对 WEP 和 IEEE 802.11i 标准中的部分数据保密技术作简单介绍。

7.2.1 WEP 算法[37]

WEP 的设计目标是对无线传输数据提供机密性、访问控制和数据完整性的安全服务。

1. WEP 的数据加/解密

WEP 对 MAC 层的数据进行序列加密，算法比较简单，如图 7.10 所示。

WEP 的密钥 k 为 40 bit(或 104 bit)，与 24 bit 的随机数初始化向量 IV(Initialization Vector)连接后输入给伪随机数发生器(采用 RC4 算法)，产生与待传输数据等长的密钥序列 ks。为了进行完整性校验，先对明文 m 计算循环冗余校验 CRC，得到 32 bit 校验码 ICV。m 与 ICV 连接后，与 ks 逐位异或即得到密文 c，算法为

$$c = \mathrm{RC4}(\mathrm{IV} \parallel k) \oplus (m \parallel \mathrm{CRC}(m)) \tag{7-12}$$

最后将密文 c 连接在 IV 后成为发送方实际发送的数据。

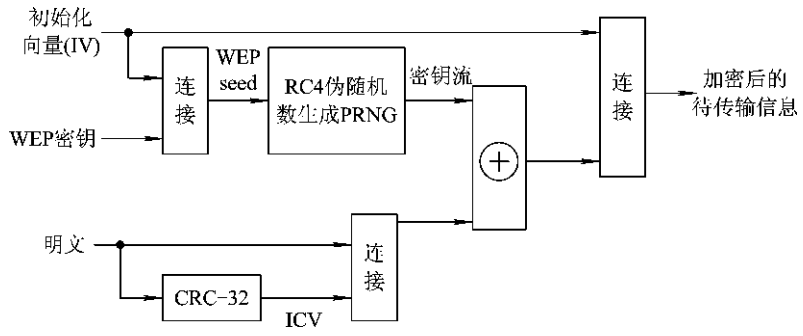

图 7.10　WEP 加密算法

因为采用序列加密算法，接收方与发送方密钥相同，解密算法与加密也是相同的。接收方收到数据信息后，从中提取 IV 和密文 c，相同的方法生成密钥流，与密文模 2 加便得到明文 m、校验和 ICV(见图 7.11)。算法为

$$\mathrm{RC4}(\mathrm{IV} \| k) \oplus c = \mathrm{RC4}(\mathrm{IV} \| k) \oplus \mathrm{RC4}(\mathrm{IV} \| k) \oplus (m \| \mathrm{CRC}(m)) = m \| \mathrm{CRC}(m) \tag{7-13}$$

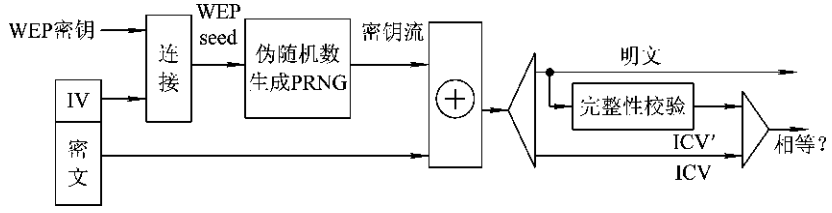

图 7.11　WEP 解密算法

为了防止数据被修改，最后仍然使用 CRC 算法对恢复的明文进行差错校验。只有计算得到的 ICV 和解密恢复的 ICV 匹配才接收该信息。CRC 算法属于纠检错编码，不展开介绍，请参考相关书籍。

WEP 的操作过程中，存在以下不足：

(1) WEP 加密中使用的 24 bit 初始向量 IV 以明文形式与加密数据一起传送。而 24 bit 的随机数共有 $2^{24}=16\ 777\ 216$ 种取值，随着发送数据包的增加，IV 会在较短时间内出现重复。

(2) 由于 WLAN 本身的特点，在无线设备上手动设置密钥后通常不再改变。

(3) 为了进行完整性校验，采用 CRC－32 计算得到校验和 ICV。但是循环冗余校验算法 CRC－32 是非加密性的线性运算，主要用于检测因突发错误而对传输数据造成的随机错码。

由于这些特点，有很多文献分析了 WEP 中存在的安全漏洞。为了改进 WEP 加密机制的各种缺陷，IEEE 组织发布了被认为是无线局域网络安全问题的最终解决方案——802.11i 标准。

2. RC4 算法

WEP 的序列密钥是用 RC4 算法产生的，现在介绍 RC4 算法。

RC4 算法是 Ron Rivest 1987 年为 RSA 数据安全公司开发的序列密码技术。RC4 算法实际上是一个序列密钥发生器，并且它并不是通过移位寄存器电路来产生伪随机密码的，

当时它享有专利。可是 7 年后，有人把它的源代码匿名张贴到网上，迅速传遍了全世界的 FTP 站点，并被拥有合法拷贝的用户确认代码无误。RSA 安全公司想要维护权益时为时已晚，因为打官司的花费并不比购买许可证少，也就放弃了。从此 RC4 就成了公开的程序，成为一种应用非常广泛的序列密码算法。

RC4 加密算法包括两个核心部分，分别是伪随机数生成算法 PRGA(Pseudo-Random Number Generator Algorithm)和密钥生成算法 KSA(Key Scheduling Algorithm)。

1) 伪随机数生成算法(PRGA)

首先创建一个 256 个字节的数组 S，并进行线性填充：$S_0=0$，$S_1=1$，…，$S_{255}=255$。同时密钥也被不断重复地填充到另一个 256 个字节的数组 K 中：K_0，K_1，…，K_{255}。然后，将两数组按以下方式进行置换：

```
j=0
For i=0, …, 255
j=(j+Si+Ki) mod 256
Swap (Si, Sj)
```

完成置换后，输入密钥就不再使用，数组 S 中就是生成的 256 个字节伪随机数。

2) 密钥生成算法(KSA)

为了生成长度可变的伪随机序列密钥，从 S 数组出发，引入两个初值为 0 的记数器 i 和 j，只需按以下程序操作，每循环一轮，就能生成一个随机字节(8 bit)：

```
i= (i+1) mod 256
j=(j+Si) mod 256
Swap (Si, Sj)
t=(Si+Sj) mod 256
Output St
```

RC4 算法十分简单，程序不过数行，软件和硬件都容易实现；而且运行速度快，大约比 DES 快 10 倍。RC4 已被广泛应用于商业密码产品中，例如 Lotus Notes、苹果机的 AQCE、Oracle 安全 SDL 数据库、移动通信的 WEP 以及安全套接字层 SSL(Secure Sockets Layer)等。

7.2.2 IEEE 802.11i

为了解决 WLAN 技术自身存在的安全上的缺陷，给无线用户提供足够的安全保护，IEEE 802.11i 任务组致力于制定新一代安全标准 802.11i。在这个标准中，为了增强 WLAN 的数据加密和认证性能，定义了健壮安全网络 RSN(Robust Security Network)的概念；在数据保密和完整性方面，定义了 TKIP(Temporal Key Integrity Protocol)和 CCMP(Counter-Mode/CBCAC Protocol)两种机制。TKIP 是 RSNA(Robust Security Network Association)的可选算法，设计初衷是通过升级固件和驱动程序来提高使用 WEP 加密机制的设备的安全性，因此是一个暂时的过渡方案。而 CCMP 机制能够提供可靠的保密性、认证性、完整性，并能抵抗重放攻击，是实现 RSN 的高级要求。但是由于它建议使用的 AES 对硬件要求比较高，CCMP 无法在现有设备基础上升级实现，因而限制了它的广泛使用。

1. TKIP 加/解密机制

TKIP 加/解密算法的核心仍然沿用了 WEP 算法，即仍然是基于 RC4 加密算法，仍使用初始向量 IV，并且要计算校验码 ICV，但对 WEP 作了很大改进。TKIP 将 WEP 密钥的长度由 40 bit 加长到 128 bit，初始化向量 IV 的长度由 24 bit 加长到 48 bit，大体相对于 802.11b 中的 WEP，其改进之处在于：

(1) 使用 Michael 算法生成 64 bit 的消息完整性认证码 MIC(Message Integrity Code)。

(2) 初始化向量 IV 用于记录序列计数器 TSC(TKIP Sequence Counter)的数值，其中的低 16 bit 参与加密密钥流的生成。这样，每个数据包有不同的加密密钥流。同时该计数器的输出值被加密在数据包的 IV 域内，以实现对 WEP 数据包格式的兼容。

(3) 为了防止 WEP 协议中出现的使用 RC4 算法弱密钥现象，TKIP 采用了混合加密函数。将临时密钥 TK(Temporal Key)、序列计数器 TSC 和初始化向量 IV 混合后生成加密器的输入包 WEP seed。

1) TKIP 加密

(1) MIC。计算消息完整性认证码 MIC 是实现 TKIP 体系的第一步。计算 MIC 的 Michael 算法要求使用 64 bit 的密钥，计算之前密钥被转换为 32 bit 的 K_0 和 K_1，输入参数有消息服务数据单元 MSDU(Message Service Data Unit)的明文数据、发送方地址 TA、目的地址 DA、优先级等，这些数据按约定的次序连接并输入到加密算法模块中后，计算得到 64 bit 的 MIC 校验码。这种完整性校验能用于防止伪造攻击。

(2) 分段(Fragment)。消息认证码 MIC 被连接在消息服务数据单元 MSDU 的数据之后，由于数据比较长，在传送时将(MSDU ‖ MIC)分割为若干个消息协议数据单元 MPDU (MAC Protocol Data Unit)。每发送出一个 MPDU，序列计数器 TSC 的值自动递增。

(3) 密钥混合函数(Key Mixing Function)。密钥混合分为两个阶段，在第一阶段，128 bit 长的临时密钥 TK(Temporal Key)、发送方的 MAC 地址 TA，以及 TSC 的高 32 bit 被送入一个 S 盒中进行运算，生成 80 bit 的中间过程密钥 TTAK (TKIP-mixed Transmit Address and Key)；在第二阶段，以中间过程密钥、临时密钥以及 TSC 的低 16 位作为输入，产生 128 bit 的 WEP 种子。加密每一个 MPDU 时，TKIP 都采用相同算法计算出不同的 WEP 种子。

(4) WEP 加密。TKIP 算法在进行 WEP 加密之前先把输入数据包 WEP seed 分解成两部分，分别作为加密器的初值 WEP IV 和 RC4 的密钥，然后把它们和消息协议数据单元 MPDU 的明文一起发送给 WEP 模块。WEP 的序列加密操作与前文所述相同，最后的输出是对 MPDU 加密后产生的密文以及校验码 ICV。加密过程如图 7.12 所示。

2) TKIP 解密

作为序列加/解密方式，TKIP 的解密过程与加密过程是类似的(见图 7.13)。

(1) 在利用 WEP 解密 MPDU 之前，需要先从 WEP IV 和扩展的 IV 中提取出 TSC 序列号和密钥 ID。为了抵抗重放攻击，如果 TSC 不符合既定排序规则，则该 MPDU 被丢弃，否则根据密钥 ID 定位 TK，并通过混合函数来创建 WEP seed。

(2)TKIP 把 WEP seed 分解成 WEP IV 和 RC4 密钥，并和 MPDU 一起送入 WEP 模块进行解密。

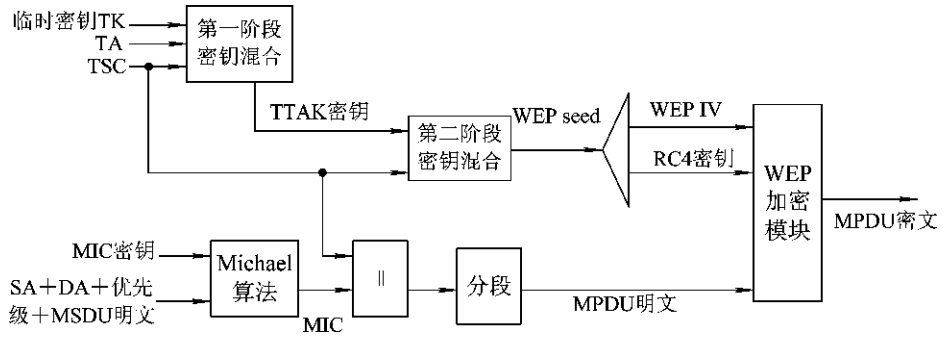

图 7.12 TKIP 加密算法

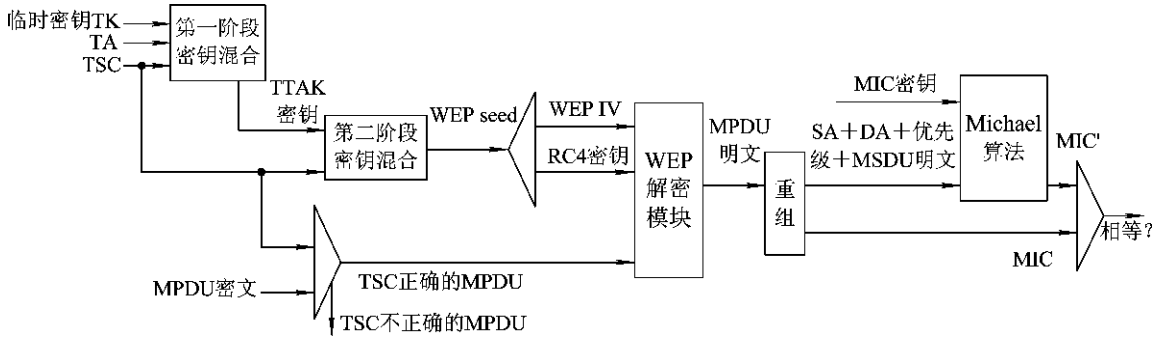

图 7.13 TKIP 解密算法

(3) **如果** WEP **检查** ICV **正确，则将该** MPDU **重组成** MSDU。**重组成功则校验** MIC **值，否则丢弃该** MSDU。

(4) **通过计算得到** MIC **后，将其与收到的** MIC **逐位比较。如果与接收到的** MICI **相同，则校验成功，**TKIP **就把** MSDU **传递给上层协议。如果不同，则校验失败，丢弃该** MSDU。

2. CCMP 加/解密机制

CCMP **机制是** IEEE 802.11i **使用的最强安全算法，基于高级加密标准** AES **和** CCM (Counter Mode with CBC-MAC)**认证方式。它是** WLAN **安全长远的解决方案，能解决** WEP **中出现的所有问题。**

CCMP **中的** AES **使用** 128 bit **密钥，它的加密块大小是** 128 bit。CCM **定义于** RFC3610，**是一种通用模式，可以和任意分组加密算法一起使用。它使用** Counter **模式提供数据保密，并采用加密块链接方式的消息认证码** CBC-MAC(Cipher Block Chaining Message Authentieation Code)**来提供数据认证和数据完整性服务。参照** 802.11i **可知，对每一个会话，**CCM **都需要一个新的临时密钥** TK **及一个唯一的随机值** Nonce，**为此** CCMP **为每个数据包引入了** 48 bit **的包号码** PN(Packet Number)。CCMP **中仍然包括** MIC，**但是不使用** WEP **的** ICV。

1) CCMP **加密**

(1) **递增的** PN **值。为了使每个** MPDU **都获得新的** PN，**在加密不同包的时候** PN **递增。即使是同样的临时密钥** TK，PN **也不会重复使用。**

(2) **构造附加鉴别数据** AAD(Additional Authentication Data)。**通过对** MPDU **头部**

(也就是 MAC 的头部)指定域的内容进行提取和重组生成附加鉴别数据 AAD。AAD 可以是 176 bit 或 224 bit。生成后再经 CCM 加密，以便为 AAD 中的数据提供完整性保护。

(3) 构造 Nonce。利用 PN、A2 和 MPDU 优先级的值计算 CCM 的 Nonce。其中 A2 是 MPDU 中地址 2 域中的内容。MPDU 优先级则置为 0，为以后的使用预留，Nonce 长为 104 bit。

(4) 构造 CCMP 头。参照 IEEE 802.11i 标准，MPDU 明文经过 CCMP 加密得到密文。密文首部要附加 CCMP 头才会被发送出去。提取 PN、密钥 ID 中指定位的数据，对包括预留位(置为 0)在内的内容进行封装、重组得到 CCMP 头。CCMP 头长度是 64 bit。

(5) CCM 加密。CCM 与 AES 分组密码联合使用进行加密生成密文和 MIC，加密操作的输入为 128 bit 的 TK、Nonce、MPDU 数据以及 AAD。通过该加密模块的运算，不仅对明文和 AAD 提供了认证性和完整性保护，还确保了明文数据的机密性。

经过上述操作，MPDU 在原基础上又扩展了 128 bit，其中 CCMP 头和 MIV 各占一半。最后，将 MAC 头部、CCMP 头部、加密数据和 MIC 连接在一起组成待发送消息。算法过程见图 7.14。

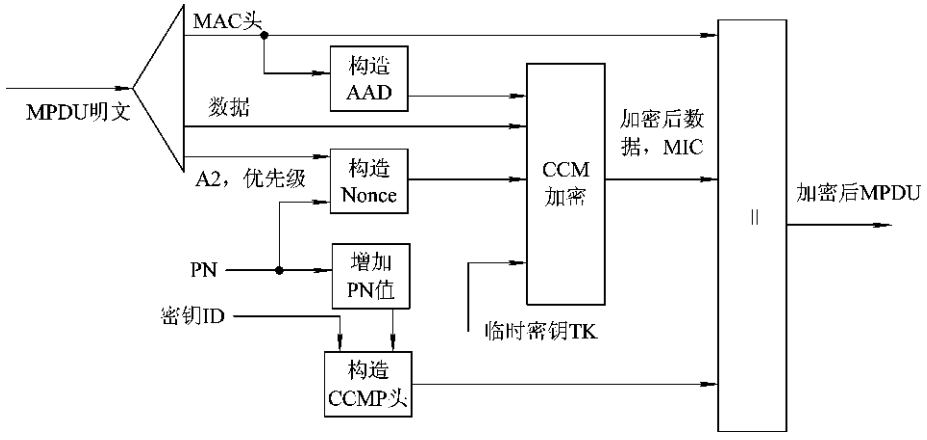

图 7.14　CCMP 加密算法

2) CCMP 解密

CCMP 解密过程如图 7.15 所示。解密过程与加密过程的方法相同，时序相反。步骤如下：

(1) 解析加密的 MPDU，获取计算 AAD 和 Nonce 值要用到的有用信息。

(2) 用与加密相同的算法，由 MAC 的头部信息重新生成 AAD。

(3) 利用 A2、PN 和优先级计算 Nonce 值。

(4) 从消息中获取用以完整性校验的 MIC 值。

(5) CCM 解密，利用 TK、AAD、Nonce、MIC 和 MPDU 的密文来恢复 MPDU 的明文，并通过校验 AAD 来验证 MPDU 明文的数据完整性。

(6) 把从 CCM 生成的 MPDU 头部和 MPDU 明文数据连接起来，即可构造出明文 MPDU。

从 CCMP 头提取 PN，并验证其数值的实时性，由此可使 CCMP 具有抗重放攻击的性能。

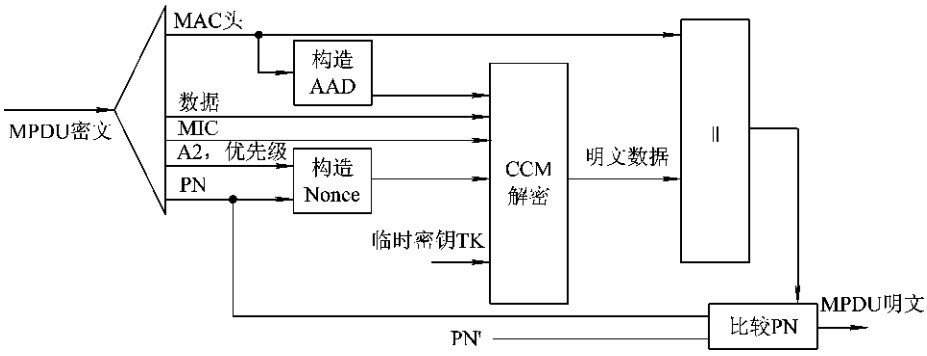

图 7.15 CCMP 解密算法

7.3 密码学在 Internet 安全技术中的应用

TCP/IP 协议是 Internet 上的主要协议族，是将不同的计算机和网络联系起来的纽带。但是它本身也存在一些安全缺陷，一方面，TCP/IP 协议的安全和控制机制只是依赖于对 IP 地址的认证；另一方面，TCP/IP 不能保证所传数据的保密性，协议包中的数据以明文的形式传送。这意味着，如果网络通信中不附加安全通信协议和机制，只依靠 TCP/IP 协议族，就不能可靠地为网络上传输的消息提供机密性、完整性和真实性等保障。

TCP/IP 协议族分为五层，如图 7.16 所示。在此基础上，已经开发和提出了许多能够提供网络安全性的方法和机制。根据在 TCP/IP 协议族的不同位置，安全机制可以分为以下三类：

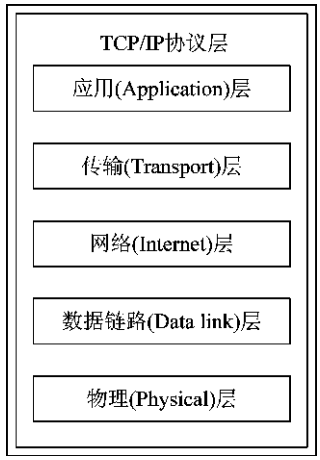

图 7.16 TCP/IP 协议分层

（1）基于网络层，保证 IP 数据报安全性的 IPSec(IP Security)协议。

（2）基于传输层，解决 TCP、UDP 传输协议数据安全问题的协议，包括 SSL(安全套接层)协议和 TLS(传输层安全)协议等。

（3）基于应用层，针对特定的网络服务(如 HTTP、电子邮件、文件传输等)，将相关安全协议直接嵌入到应用程序中的安全机制。例如针对 HTTP 的 SHTTP(Secure HTTP)协议；针对安全电子邮件的 PGP、S/MIME 协议；针对安全信用卡交易的 SET(安全电子交易)协议等。

这些安全机制中都大量使用了对称加密、公钥加密、数字签名及身份认证等现代密码算法。本节就几种典型机制：IPSec、SSL 以及面向电子邮件安全的 PGP 和 S/MIME，介绍其密码算法及其应用。

7.3.1 IP 安全协议(IPSec)[39]

1994 年，IETF 成立了一个 Internet 安全协议工作组，负责 IP 安全协议和密钥管理机制的制定。经过多年工作，工作组提出了一系列 IP 安全协议标准及相关扩展，这些协议总

称为 IP Security 协议，简称 IPSec。IPSec 的基本架构文档是 RFC2401[40]，其中定义了 IPSec 基本结构，所有具体的实施方案均建立在此基础上。IPSec 主要包括鉴别头 AH (Authentication Header)协议、封装安全载荷 ESP(Encapsulating Security Payload)协议，以及 Internet 密钥交换 IKE(Internet Key Exchange)协议。基于密码学等技术，IPSec 提供了一种标准的、包容广泛的安全机制，用它可以为 IP 层及上层协议提供安全服务，从而支持 IP 数据报的认证、完整性和机密性。同时这些机制均独立于具体算法，所以具体实现时系统可在 IPSec 定义的范围内选择合适的安全策略，灵活决定所需的算法和密钥。

图 7.17　IPSec 在 TCP/IP 协议层

IPSec 在 TCP/IP 协议分层中的位置如图 7.17 所示。

IPSec 协议有两种操作模式，传输模式和隧道模式。传输模式下，保护的是 IP 层载荷，在 IP 头与上层协议头(如 TCP、UDP)之间插入安全协议(AH/ESP)头，当要求两个主机端到端进行安全通信时，通常使用传输模式。隧道模式下，要保护的整个 IP 包都被封装到一个新的 IP 数据报中，即加上一个新的外层 IP 头，安全协议(AH 或 ESP)头介于外层和内层 IP 头之间。隧道模式通常用于有安全网关(防火墙或路由器)的网络配置中。两个安全协议 AH 和 ESP 都支持这两种工作模式。

1. 基本协议 AH 和 ESP

AH 和 ESP 可以单独使用，AH 主要负责认证，ESP 主要负责保密。它们也可以一起使用，一起使用时把加密和认证结合起来，在主机之间传输具有认证和机密性的 IP 包。

1) AH

RFC2402[41]对鉴别头 AH 进行了定义。AH 用于为 IP 包提供数据完整性、数据源身份认证等服务，能防止传输过程中对数据包内容的修改、地址欺骗攻击以及信息重放攻击等。AH 头格式如图 7.18 所示，其中：

下一个头 (8 bit)	载荷长度 (8 bit)	保留 (16 bit)
安全参数索引(SPI 32 bit)		
序列号(SN 32 bit)		
认证数据(长度可变)		

图 7.18　AH 封装格式

- 下一个头用来描述 AH 后面的头类型。
- 载荷长度表示认证数据的长度，它以 32 bit 为一个字，其值等于字数减 2。
- 安全参数索引用以标识安全关联。
- 序列号是一个单调递增的计数器数值，用来防止 IP 包重放攻击。
- 认证数据域是一个变长域，包含了数据报文的鉴定数据，称为完整性校验值(ICV)，是 AH 的核心。ICV 是将传输过程中变化的域和认证数据域置零后，对其余所有数据通过鉴别算法计算后得到的。其长度必须是 32 位字长的整数倍，如果不满足这个条件，则在数据后添加填充字段。AH 中默认的鉴别是消息鉴别码 MAC，当前规定的支持算法是 HMAC－MD5－96 和 HMAC－SHA－1－96。经过 HMAC 计算，前者产生 128 bit 的散列值，后者产生 160 bit 的散列值，均选取其高 96 bit(认证数据的默认长度)作为校验值 ICV。

2）ESP

RFC2406[42]对 ESP 进行了定义。ESP 用于为 IP 数据报提供保密性和抗重放服务。通过对数据包的全部数据和加载内容进行全加密来保证传输信息的机密性，使只有拥有密钥的用户才能解密内容。由于 ESP 实际上加密了所有的数据，因而它比 AH 需要更多的处理时间，从而导致性能下降。ESP 头格式见图 7.19。其中的认证数据是 ICV，它对除身份验证数据本身之外的整个 ESP 头进行计算。

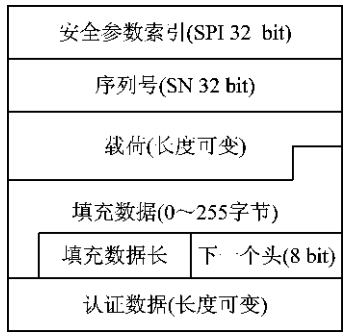

图 7.19 ESP 封装格式

作为可选功能，ESP 也可以提供和 AH 相同的数据完整性和鉴别服务，但是在具体实现上和 AH 稍有不同。AH 对外部 IP 头各部分都进行身份鉴别，以此检验数据的完整性；而 ESP 并不鉴别外部 IP 头，只对外部 IP 头之后的内容进行身份鉴别。ESP 目前要求必须支持加密块链 CBC (Cipher Block Chaining)模式的 DES 加密算法。在解释域，文档还对 ESP 支持的很多其他算法分配了标识符，可以方便地使用它们。这些算法包括：三重 DES、IDEA、RC5、CAST、Blowfish 等。ESP 最主要的加密算法是数据加密标准(DES)，DES 最高支持 56 bit 的密钥，而 3DES 使用 3 套密钥加密，相当于使用最高到 168 bit 的密钥。

2. 密钥管理[43]

除了前述两个核心协议 AH 和 ESP 之外，IPSec 的另一个重要组成是密钥管理。在密钥管理过程中，IPSec 协议使用 Internet 密钥交换协议(IKE)。AH 的鉴别算法和 ESP 加密算法都需要密钥，若同时使用两协议则需要的密钥更多，为 4 个，即各一对发送/接收密钥。

在 IPSec 中进行密钥交换有两种方法：一种是使用 IKE 协议进行自动密钥交换；一种是手工模式。手工模式只适用于小规模的或者用硬件实现的 IPSec，大多数情况下都需要使用 IKE 协议通过公用网络进行密钥交换。

Internet 安全关联和密钥管理协议 ISAKMP 规定了一个认证和密钥交换的框架，定义了所有用于建立安全关联和执行密钥管理功能的协议结构，并且与具体的密钥交换方法、加密、鉴别算法相独立。Internet 密钥交换协议 IKE 使用 ISAKMP 语言来定义密钥交换，因此是 ISAKMP 框架的实例化。IKE 的功能包括加密算法、密钥协商、密钥生成、密钥交换及管理等，现已成为主要的密钥管理标准。

IKE 使用了两个阶段的 ISAKMP。第一阶段建立 IKE 安全联盟(IKE SA)，为下一步的 IKE 通信提供保密性、消息完整性及信源验证服务；第二阶段利用这个初步的安全联盟，为 IPSec 协商具体的安全联盟(IPSec SA)，并为安全联盟的属性进行定义。

IKE 是在 Diffie－Hellman 密钥交换协议的基础上，经过改进制定的。改进之处主要表现在：

(1) 使用 Cookie 机制防止拥塞攻击。通信双方首次所发送消息中携带一个伪随机数，称为 Cookie，对方必须在每次交换 Diffie－Hellman 密钥之前确认该 Cookie。如果攻击者伪造源地址，则不能得到对方确认的 Cookie。

(2) 支持三种身份认证机制，分别是数字签名(生成消息摘要，并用双方各自私钥加

密)、公钥加密(用发送方私钥加密)以及对称密钥加密(使用双方事先共享的密钥和对称加密算法加密)。

(3) 通信双方可交换 Diffie-Hellman 公开密钥值。IKE 定义了 5 个 Diffie - Hellman 组，支持在不同的组中完成 Diffie - Hellman 密钥交换，其中 3 个组使用模幂算法(模分别是 768、1024、1680 bit 的大素数)，另两个组使用了椭圆曲线算法(分别定义于 GF[2^{155}]和 GF[2^{185}]域)。

(4) 使用表征密钥生成时间的伪随机数 nonce，并在交换协议中加密以防止重放攻击。

总之，IPSec 并不限定某种特定的加密和散列算法，它提供的是一种让设备协商的平台，采用构建安全关联(SA)的方案来确定双方使用的通用策略和会话密钥。通过 SA 通信，各方协商确定使用的 IPSec 版本、操作方式(隧道模式或传输模式)、加密算法、加密密钥等。IKE 交换的最终结果是双方得到了一个通过验证的密钥，建立了双方一致同意的安全服务 IPSec 安全联盟(IPSec SA)。

7.3.2　安全套接层(SSL)协议[20, 21]

安全套接层(Security Socket Layer，简称 SSL)协议由 Netscape 公司开发，是目前 Internet 上使用最广泛的安全协议，几乎所有的 Web 浏览器都支持 SSL。它借助 PKI(公钥基础设施)的保障机制，有效地保证了信息交换的安全。SSL 协议制定了一种在应用程序协议(如 HTTP、Telenet、NMTP 和 FTP 等)和 TCP/IP 协议之间提供数据安全分层的机制。SSL 位于 TCP 层和应用层(程序)之间，其位置以及协议分层和包含的子协议如图 7.20 所示。SSL 第 3 版是最常用的，已被开发为一般标准。

图 7.20　SSL 在 TCP/IP 协议层中的位置

SSL 在 Web 浏览器和 Web 服务器之间主要提供三方面的服务：

(1) 客户端和服务器的合法性认证：通信双方利用数字证书或非对称加密算法完成双向认证或单向身份认证。其算法采用 RSA、DSS 等。

(2) 采用加密技术隐藏被传送的数据：通信双方经过握手协议确定加密算法和私有密钥，加密传输数据。数据加密采用对称加密算法(如 DES、RC6 等)。

(3) 保护传输数据的完整性：采用消息鉴别码(MAC)对数据的完整性进行验证。MAC 依靠 Hash 函数(如 SHA、MD5 等)来实现。

1. SSL 原理

SSL 的连接分为两个阶段，即握手阶段和数据传输阶段。

1) 握手协议

SSL 通过握手协议来实现握手功能，使服务器和客户端双方得以协商和交换相关版本号、压缩方法、加密算法和密钥等信息，完成客户方对服务器方的认证(服务器方对客户方的认证为可选)，同时使用公钥技术来生成会话密钥(Session Key)。

握手协议的工作流程为

(1) 客户端向服务器发送 ClientHello 消息，服务器以 ServerHello 响应。这个过程中交换和确定了包括协议版本、会话标识、加密算法、压缩方法等参数。另外还交换了两个随机数 ClientHello. random 和 ServerHello. random，用于计算共享密钥。

(2) 服务器发送包含公钥参数的证书给客户端。如果有必要，则还要求发送客户端证书。最后发送 ServerHelloDone 表示这一阶段已完成。

(3) 客户端对服务器的证书进行验证，抽取服务器的公钥，并根据服务器的相关请求进行响应。之后，产生随机密码串 pre-master-secret，并用证书上的公钥加密后发送给服务器。

(4) 服务器解密得到 pre-master-secret。双方根据 pre-master-secret 以及交换的随机数独立计算出加密密钥和 MAC 密钥。

(5) 客户端与服务器各自使用协商好的算法和 MAC 密钥加密所有握手消息，并发送给对方，形成 Finished 消息，完成对密钥交换和认证过程的正确性验证。

2) 记录协议

SSL 中实际的数据传输是通过 SSL 记录协议来实现的。发送方的记录层收到上层待传输数据后，首先将数据分块，每个块的大小不得超过 2^{14} 字节，接着对每个数据块进行压缩(该操作可选)。为防止信息在传输中被篡改，按照握手协商的散列算法添加消息鉴别码(MAC)。再对压缩后的数据连同 MAC 一起用对称加密算法加密形成负载。给负载装上记录头，记录头与经过加密的负载的连接称做记录，最后将记录发送给接收方。

接收方的记录层收到消息后首先对加密的负载解密，然后计算 MAC，对数据进行验证。验证通过后利用解压缩算法恢复出原始消息。

2. SSL 中使用的算法

SSL 中使用的加密算法有对称密钥加密、公开密钥加密和消息摘要三大类。

对称密钥加密用于加密应用数据。算法有 DES、IDEA、RC2、3DES 等。

公开密钥加密用于验证实体和交换密钥，按用途分为密钥交换算法和数字签名算法。密钥交换算法有 RSA、Diffie-Hellman 等；数字签名算法有 RSA、DSS。

数字签名用于完整性认证和信源认证。用单向散列值作为签名算法的输入时，RSA 签名中的散列算法是 SHA-1 和 MD5，密钥分别是 128 bit 和 160 bit；DSS 签名中的散列算法是 SHA-1，密钥是 160 bit。

1) SSL 握手协议中的 Hash 算法和签名

SSL 第 3 版对 Hash 算法使用得很谨慎，在可能的情况下，MD5 和 SHA 一前一后地使用(或相继使用，或嵌套使用)，这就保证了一个 Hash 算法被攻破时不至于导致整个协议被攻破。在握手协议第(2)步中，服务器可能需要签名，通常是对消息的 Hash 函数值签名，算法可以使用 RSA 或 DSS。

Hash 函数的运算形式为

$$\text{hash}(\text{ClientHello.random} \parallel \text{ServerHello.random} \parallel \text{ServerParams}) \tag{7-14}$$

其中，ClientHello. random 和 ServerHello. random 是初始 hello 消息中的两个随机数；ServerParams 是双方协商的参数。DSS 签名的 Hash 函数使用 SHA-1，对 20 字节的散列值直接签名；RSA 签名的 Hash 函数使用 MD5 和 SHA-1，将两个散列结果连接，再将得

到的 36 字节数据用服务器私钥签名。

握手协议最后一步的 Finished 消息中，客户端与服务器用 HMAC 对过程的正确性进行验证，密钥是 master_secret。验证内容是下面两个 MAC 结果的连接：

MD5(master_secret ‖ pad_2 ‖ MD5(handshake_messages ‖ Sender ‖ master_secret ‖ pad_1))；

SHA(master_secret ‖ pad_2 ‖ SHA(handshake_messages ‖ Sender ‖ master_secret ‖ pad_1))；

(7-15)

其中，pad_1 定义为字节 0x36，对 MD5 重复 48 次，对 SHA-1 重复 40 次；pad_2 定义为字节 0x5C，对 MD5 重复 48 次，对 SHA-1 重复 40 次；handshake_messages 指除此部分之外的所有握手消息数据；master_secret 称为主密钥，用它来生成其他密钥，进行加密以及进行 MAC 计算。

生成主密钥的计算方法为

$$\begin{aligned}\text{master\_secret}=&\text{MD5}(\text{pre\_master\_secret}\,\|\,\text{SHA}('\text{A}'\,\|\,\text{pre\_master\_secret}\,\|\\&\quad\text{ClientHello.random}\,\|\,\text{ServerHello.random}))\,\|\\&\text{MD5}(\text{pre\_master\_secret}\,\|\,\text{SHA}('\text{BB}'\,\|\,\text{pre\_master\_secret}\,\|\\&\quad\text{ClientHello.random}\,\|\,\text{ServerHello.random}))\,\|\\&\text{MD5}(\text{pre\_master\_secret}\,\|\,\text{SHA}('\text{CCC}'\,\|\,\text{pre\_master\_secret}\,\|\\&\quad\text{ClientHello.random}\,\|\,\text{ServerHello.random}))\end{aligned}\tag{7-16}$$

其中，A、BB 和 CCC 是实际的 ASCII 值。

SSL 连接中的 MAC 密钥和对称加/解密数据所用的写密钥由密钥块 key_block 得到。密钥块的计算如下：

$$\begin{aligned}\text{key\_block}=&\text{MD5}(\text{master\_secret}\,\|\,\text{SHA}('\text{A}'\,\|\,\text{master\_secret}\,\|\,\text{ServerHello.random}\,\|\\&\quad\text{ClientHello.random}))\,\|\\&\text{MD5}(\text{master\_secret}\,\|\,\text{SHA}('\text{BB}'\,\|\,\text{master\_secret}\,\|\,\text{ServerHello.random}\,\|\\&\quad\text{ClientHello.random}))\,\|\\&\text{MD5}(\text{master\_secret}\,\|\,\text{SHA}('\text{CCC}'\,\|\,\text{master\_secret}\,\|\,\text{ServerHello.random}\,\|\\&\quad\text{ClientHello.random}))\,\|\cdots\end{aligned}\tag{7-17}$$

直至生成足够的密钥块，然后按照握手协议第(1)步中商定的方法抽取获得 MAC 密钥和对称写密钥。

2) SSL 记录协议中的对称算法

为了保证记录协议层传输数据的保密性，对压缩后的消息连同其消息鉴别码(MAC)要用对称加密算法加密。SSL 第 3 版允许使用分组密码或序列密码，其中主要有如表 7.1 所例的算法。

流加密是将压缩的报文与 MAC 相连接后直接进行序列加密。分组加密由于分组密钥要求明文长度为分组长度的整数倍，因此在 MAC 之后可以增加填充。填充是由表示填充长度的字节跟着一定数目的填充字节组成的，填充字节的数目要使得被加密的数据(即明文加上 MAC，再加上填充)的总长度成为加密分组长度的最小整数倍。

表 7.1 SSL 记录协议中的对称算法

分组密码		序列密码	
算法	密钥大小/bit	算法	密钥大小/bit
IDEA	128	RC4	40、128
RC2	40		
DES	40、56		
3DES	168		
Fortezza	80		

7.3.3 电子邮件安全技术[20,21]

随着电子邮件的广泛使用，邮件认证和保密的需求也日益增长。目前在 Internet 上有两个较为完善的端到端安全电子邮件标准，它们是 PGP(Pretty Good Privacy)和安全多功能 Internet 邮件扩展 S/ MIME (Secure Multi—Purpose Internet Mail Extension)。

1. PGP

PGP 是一个在学术界和技术界得到广泛认可的安全邮件标准，创始人是美国的 Phil Zimmermann。PGP 并非是单纯的加密算法，而是一种公钥体系和其他对称加密体系相结合的混合加密系统，其中使用了公开密钥算法 RSA、分组加密算法 IDEA 以及散列算法 MD5 等。PGP 充分利用这两类加密算法的特性，实现了鉴别、加密、压缩等多重功能。目前常见的软件 PGP 是由美国 Networks Associates Technology 公司开发的，它是第一个使得公钥密码服务于大众的工具，主要用来加密电子邮件，现在也用来加密文件和实现 VPN。它适用于多种平台和操作系统。

1) PGP 原理及密码的应用

PGP 处理邮件的步骤为数字签名、压缩、加密与编码变换。操作中用到的各种密码算法如图 7.21 所示。

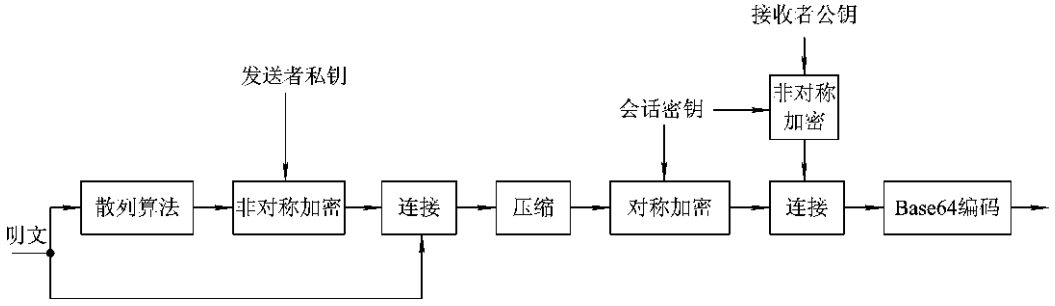

图 7.21 PGP 发送操作中的算法

(1) 数字签名。发送者创建邮件，并使用 SHA－1 等散列算法生成消息摘要，随后使用自己的私钥采用 DSS 或 RSA 对摘要加密，形成发送方的数字签名。

接收者使用发送者的公钥，采用 RSA 解密得到消息摘要，然后与根据接收到的报文重新计算的散列代码比较，如果匹配，则接收报文。

数字签名能够保证接收者接收的信息没有经过未授权的第三方篡改，并确信报文来自发信者。目前，PGP 使用的散列函数有 SHA－2(256 bit、384 bit、512 bit)、SHA－1(160 bit)、MD5(128 bit)等。

(2) 压缩。压缩邮件是为了减少网络传输时间和磁盘空间，并减少明文上下文间的关联，提高安全性。压缩发生在签名之后、加密之前。PGP 使用的是 ZIP 压缩算法。

(3) 加密。发送方首先产生适当长度的随机数，用它作为一次性会话密钥对压缩后的文件进行加密。之所以叫做一次性会话密钥，是因为它是收、发双方秘密通信所必须的共同的密钥，而且为了安全起见，发信人每发送一个消息都使用一个不同的密钥。加密采用的是对称加密算法，包括 AES、CAST－128、3DES、IDEA 等。PGP 使用对称密钥加密是出于时间上的考虑，对称加密算法比公钥加密速度快大约 10 000 倍。

邮件加密之后，还要使用接收者的公钥来加密会话密钥，可采用的公钥加密算法是 Diffie-Hellman、RSA 或 ElGamal。使用公钥加密来封装会话密钥是为了解决会话密钥的交换问题，让收信人能够安全地得到本次通信的一次性会话密钥。

最后将密文与加密后的一次性密钥一起发出。接收方则先用私有密钥解密，获得一次性会话密钥，再用这个密钥解密密文。显然只有拥有私钥的合法接收者才能解译所发的信件。

(4) 编码变换。几乎所有的 E-mail 程序都支持 Base64 与 UUEncode 这两种编码方式。PGP 采用 Base64 编码，是为了与其他 E-mail 软件相兼容。Base64 编码是将所有的 8 bit 二进制码统统变换成 64 个常用可显示字符(26 个大写字母，26 个小写字母，10 个自然数以及“＋”和“/”)的一种编码。因为经过压缩与加密处理的 PGP 密文，其中必然包含各种二进制码的组合，如果不进行变换，遇到许多邮件收发程序时，就难以识别或者发生错误。

Base64 编码的原理是将 3 个字节的 24 bit 数据依次装入 4 个 6 bit 的数据单元中(见示意图 7.22)，于是每个 6 bit 数据就可以对应 64 个常见字符中的一个字符。程序细节可参见文献[45]。

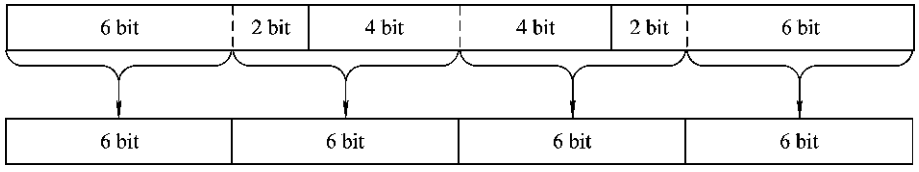

图 7.22　Base64 编码原理示意

2) PGP 密钥管理

在 PGP 里面，最有特色的或许就是它的密钥管理。PGP 包含四种密钥：一次性会话密钥、公开密钥、私有密钥和基于对称密钥的口令短语。

PGP 采用一次一密的方式，每一次发送消息都需要一个不同的密钥加密。产生会话密钥的算法可以是 CAST－128、IDEA 或 3DES 等。由于产生的随机数是不可预测的，且每次不同，因此只需要保留密钥发生器，不需要保管一次性会话密钥。

用户使用 PGP 时，必须备有一个公钥/私钥对(其中公钥可以公开)。PGP 用两个文件存储密钥，一个用来存储本用户的公/私钥，称为私钥环；另一个用来存储其他用户的公钥，称为公钥环。

为了确保只有本用户可以拿到私有密钥，PGP 采用了有效的保密措施。当用户生成一

个新的公开/私有密钥对时，立即被要求输入一个口令短语，并用散列算法(如 SHA－1)计算出它的 Hash 码，将其作为密钥对用户的私钥加密，然后存储在私钥环中。每当用户需要提取私钥时，必须提供相应的口令短语，根据口令短语和相应算法，获得 Hash 码，才能解密得到私钥，从而保证了私钥的安全性。

至于 PGP 对公钥的管理，PGP 没有建立认证中心，而是采用信任机制。PGP 公钥环不应当只是一张记载与本用户有通信关系的其他用户公钥的地址簿，还应该是一个公钥证书(Public Key Certificate)的数据库。公钥环上的每一项记录都应当包含捆绑了用户公钥的身份信息和相关联的一个或多个签名。此外，每个记录都有一个密钥合法性字段，表示此公钥的真实性，由 PGP 计算得到。而且每个签名都应带有一个信任等级字段，用来指示对这个公钥证书的信任程度。信任级别包括不了解(unknown)、不信任(untrusted)、部分信任(marginally trusted)和完全信任(completely trusted)四个等级。密钥合法性字段是从该项的一组签名信任字段中推导出来的。每当向公钥环中插入一个新公钥时，都要根据上面附加的签名来计算信任值的加权值，确定该密钥的合法程度。

2. 多用途互联网电子邮件扩展(S/MIME)[46]

多用途互联网电子邮件扩展 MIME (Multipurpose Internet Mail Extensions)标准的建立，使得原本无格式的文本信件可以含有多个部件和多种结构，并且每一个部件都可以有不同的信息编码类型，并能传送非文本信息。MIME 常用来传输邮件附件，它也支持国际字符集的使用，支持 HTML 格式，支持音频、语音邮件、图像、视频等。安全的多功能因特网邮件扩展 S/ MIME (Secure Multi-Purpose Internet Mail Extension)是在 MIME 基础上添加数字签名和加密技术的一种协议，已获得广泛认可。IETF 提议使用 S/MIME 版本 3，已有众多软件厂商把它作为安全邮件的标准。它支持绝大多数的流行加密算法，并能很好地保密全部的 MIME 信息，可以被大部分基于 MIME 的邮件软件识别和处理。

与 PGP 比较而言，S/MIME 的认证基于 X.509 版本 3 的标准，但采用分级的多认证机构形式，类似树状的信任关系，要求所有下一级的组织和个人的证书由上一级的组织负责认证，而最上一级的组织(根证书)之间相互认证。

尽管 PGP 和 S/MIME 都是 IETF 的标准，但是 S/MIME 可作为商业和组织使用的工业标准而出现，而 PGP 主要用于个人安全电子邮件。

1) S/MIME 的功能

从功能上来说，S/MIME 与 PGP 非常类似，两者都提供了对报文签名和(或)加密的功能，然而 S/MIME 还提供了封装数据、签名数据、透明签名(Clear-Signed)的数据、签名并封装数据等功能。

封装数据：支持保密性服务。用对称密钥加密消息中的任何类型的内容，然后用一个或多个接收者的公钥加密对称密钥。并将密文和加密后的对称密钥及任何必要的接收者标识符和算法标识符等一起附加在数据后面。

签名数据：提供完整性服务。发送者能够对选定的内容计算消息摘要，然后用签名者的私钥加密，并将原始内容及相应签名都进行 Base64 编码。因此一个签名后的消息只能被有 S/MIME 功能的接收者处理。

透明签名数据：对选定的内容计算消息摘要，形成数字签名，并进行 Base64 编码，而其他内容不变。因此没有 S/MIME 功能的接收者无法验证签名，但可以看到消息的内容。

签名并封装数据：可以组合签名与封装技术，签名已加密的数据或者加密已签名(或透明签名)的数据，同时提供保密性和完整性服务。

2) S/MIME 消息处理过程

S/MIME 消息处理过程中消息摘要的生成、数字签名、对称密钥加密以及封装等都与 PGP 相似，这里不再赘述。重要区别在于 S/MIME 消息的准备过程。

S/MIME 准备过程首先安全化一个 MIME 实体，然后按照 S/MIME 内容类型来包装数据。S/MIME 使用签名、加密或同时使用两者来保证 MIME 实体的安全。一个 MIME 实体可能是一个完整的消息(除了 RFC822 规定的消息首部)，或者当 MIME 内容类型是多部分时，MIME 实体是一个或多个消息的子部分。MIME 实体按照 MIME 消息准备规则来准备。然后，该 MIME 实体加上一些与安全有关的数据，如算法标识符和证书，被 S/MIME 处理以得到公钥加密标准 PKCS(Public Key Cryptography Specification)对象。最后 PKCS 对象被作为消息内容封装到 MIME 中。

与 PGP 类似，在消息发送之前要进行编码转换，一般仍然使用 Base64 编码。

3) S/MIME 中的密码算法

S/MIME 使用了多种密码算法：

(1) 用于数字签名的散列函数，S/MIME 推荐使用 160 bit 的 SHA－1，但为了与 S/MIME 版本 2 兼容，要求接收方同时还支持 128 bit 的 MD5。

(2) 数字签名标准(DSS)是用于数字签名的推荐算法。

(3) Diffie-Hellman 是用于加密会话密钥的推荐算法。实际上 S/MIME 使用的是加密算法 ElGamal。RSA 作为候选算法，既可用于签名，也可用于加密会话密钥。

(4) 对于消息加密，推荐使用 3DES，但是实现时也应支持 40 bit 的弱加密算法 RC2。

表 7.2 总结出了 S/MIME 中加密算法的使用情况。

表 7.2　S/MIME 中使用的加密算法

功　能	要　求
创建用于形成数字签名的消息摘要算法	发送和接收代理必须支持 SHA－1 接收代理应该支持 MD5 以便与以前版本兼容
加密消息摘要以形成数字签名	发送和接收代理必须支持 DSS 接收代理应该支持 RSA 发送代理应该支持 RSA，发出的消息使用长度为 512～1024 bit 的密钥来签名
加密会话密钥和消息一起传送	发送和接收代理必须支持 Diffie-Hellman 接收代理应该支持 RSA，对收到的已加密会话密钥使用长度为 512～1024 bit 的 RSA 解密 发送代理应该支持 RSA
使用一次性会话密钥加密消息	发送代理应该支持 3DES 和密钥长度为 40 bit 的 RC2 接收代理必须支持 3DES，应该支持密钥长度为 40 bit 的 RC2

根据 RFC 文档的统一要求，表 7.2 中 S/MIME 使用了两个术语来说明要求的级别，含义如下：

- MUST(必须)：该规定是实现协议的必须要求。
- SHOULD(应该)：或许能忽略，但是推荐实现时包括该项。

习 题 7

1. KASUMI 算法与 Rijndael 密码(见本书第 4 章)有何异同点？

2. 3G 中 $f8$ 算法和 $f9$ 算法的主要作用是什么？

3. 试分析 WEP 协议中如果密钥序列重复会出现什么安全漏洞？

4. IPSec 中 ISAKMP 和 Oakley 协议的作用是什么？

5. 考虑以下 Web 安全性威胁，并描述 SSL 是如何防止这些威胁的：

(1) 穷举密码分析攻击：穷举传统加密算法的密钥空间。

(2) 重放攻击：重放先前的 SSL 握手消息。

(3) 中间人攻击：在密钥交换时，攻击者向服务器冒充客户端，向客户端冒充服务器。

(4) 密码窃听：窃听如 HTTP 等应用的密码。

(5) IP 欺骗：使用伪造的 IP 地址使主机接收伪造的数据报。

6. PGP 使用了密文反馈模式(CFB)，而传统的加密多使用密文块的连接模式(CBC)(见本书第 4 章)。分别表示如下：

CBC：$c_i = E_K[c_{i-1} \oplus m_i]$，$m_i = c_{i-1} \oplus E_K[c_i]$

CFB：$c_i = P_i \oplus E_K[c_{i-1}]$，$m_i = c_i \oplus E_K[c_{i-1}]$

这两种模式看起来提供了相当的安全性。请问 PGP 使用 CFB 的原因是什么？

实 践 练 习 7

实验题目： IPSec 协议与 IPSec 的安全服务。

实验平台： Windows 2000 及以上版本。

实验内容： 配置 IPSec 安全策略，禁止某些协议，关闭不安全端口，提高通信的安全性。

实验步骤：

(1) 熟悉 IPSec 基本概念及 AH 和 ESP 异同点。

(2) 创建和应用 IPSec 策略，达到禁止协议和关闭端口的目的。

(3) 创建和应用 IPSec 策略，要求通信双方在通信过程中必须进行身份验证，且对发送数据使用完整性算法和加密算法。

· 配置提示

1) 实验步骤 2

禁止协议：

(1) 在 Windows“开始”菜单中选择“运行”，键入“mmc”，启动管理控制台，在“文件”菜单中打开“添加/删除管理单元”对话框，添加“IP 安全策略管理”管理单元。创建一个新的 IP 安全策略。根据向导，点击“下一步”，在“IP 策略名称中”命名后，不启用“激活默认响应规则”，在点击完成时选择“编辑属性”，开始编辑这一策略的属性。

（2）打开属性界面，单击“添加”，在“IP 筛选器列表”中选择“添加”；在“筛选器”的“寻址”中选择筛选器的方向；在“源地址”中选择“任何 IP 地址”；在“目标地址”中选择“我的 IP 地址”；选择“镜像”。在“协议”中选择协议类型并“确定”。

（3）在“新规则属性”中的“IP 筛选器列表”选择该 IP 筛选器。在“常规”里命名后，在“安全措施”里选择“阻止”即可。

关闭端口：

（1）进入控制台，创建一个新的 IP 安全策略。根据向导，点击“下一步”，在“IP 策略名称中”命名后，不启用“激活默认响应规则”，在点击完成时选择“编辑属性”，开始编辑这一策略的属性。

（2）在“IP 筛选器列表”中选择“添加”，并命名。单击“添加”，在下一个“筛选器属性”对话框中的“寻址”一栏的源地址里选择“任何 IP 地址”；在目标地址中选择“我的 IP 地址”。在“筛选器属性”对话框的“协议”一栏中，找到“选择协议类型”选项，选择所需要的协议。

（3）在 “设置 IP 协议端口”中选择“从任意端口”和“到此端口”。

（4）在“筛选器操作”选项中，选择相应阻止或通过对应的筛选器操作名称，然后选择“应用”和“确定”，右键点击“指派”。

2）实验步骤 3

（1）打开待配置策略属性对话框。可以在已有的 IP 筛选器上继续配置。

（2）在“新规则属性”的“筛选器操作”中，单击“添加”→“属性”→“安全措施”，选择“协商安全”。选择“添加”，出现“新增安全措施”的对话框。选择“加密并保持完整性”，单击“自定义”→“设置”，选择相应完整性计算的算法及加密算法，并设置会话密钥参数。

（3）在“新规则属性”对话框中设置身份验证的参数。可选择默认身份验证方法“Kerberos” 或根据证书颁发机构颁发的许可证书，以及第三种设置预共享密钥。

（4）在通信对端的计算机上设置同样的 IPSec 策略。

附录 A 数学补充知识

A.1 因式分解与模运算

A.1.1 定义和记号

首先声明，以下定义和记号中所出现的数，均为整数。

(1) 整除：b 能被 a 整除，记作 $a|b$，b 称作 a 的倍数，a 称作 b 的因数。

(2) 素数：只能被 1 和自身整除的整数。

(3) 合数：除了 1 和自身，还可以被其他整数整除的整数。

(4) 公因子：若 $a|b$ 且 $a|c$，则 a 是 b 和 c 的公因数。

(5) 最大公因子：记作 $a=(b, c)$ 或 $a=\gcd\{b, c\}$，是 b 和 c 的公因子中最大的一个。

(6) 最大公因数存在定理：每一对不为 0 的整数，必有一个最大公因数。

(7) 公倍数：若 $a|c$ 且 $b|c$，则 c 是 a 和 b 的公倍数。

(8) 最小公倍数：记作 $c=[a, b]$ 或 $\mathrm{lcm}\{a, b\}$，是 a 和 b 的公倍数中最小的一个。

(9) 互素：若 $\gcd\{a, b\}=1$，则称 a 与 b 互素。a 和 b 不一定是素数，但它们没有公因子。

(10) 整商：$p=\left[\frac{x}{m}\right]$ 表示 x 除以 m 的商的整数部分；记号 $[y]$ 表示不大于实数 y 的最大整数。

(11) 余数：$q=x \bmod m$ 表示 x 除以 m 后剩余的整数。这时，$x=pm+q$。

(12) 模运算：求余运算(mod)也叫做求模运算。经常见到的形式为

$$y=f(x) \bmod m$$

它表示等式 $y=f(x)$ 是在模 m 运算下成立的，并非只是对等式右边求余。

A.1.2 m 进制整数表示法

常用十进制数表示整数，例如 1987 的意义为

$$1987=1\times10^3+9\times10^2+8\times10^1+7\times10^0$$

若用二进制表示这个数，则只需将基底换成 2 即可：

$$1987=2^{10}+2^9+2^8+2^7+2^6+2^1+2^0=11111000011\mathrm{B}$$

定理：任何正整数 n_0 都可以唯一地表示为 $m(m>1)$ 进制数：

$$n_0 = C_k m^k + C_{k-1} m^{k-1} + \cdots + C_1 m + C_0 \tag{A-1}$$

其中：C_0 **是** n_0 **除以** m **的余数**，$C_0 = n_0 \bmod m$，**可写为**

$$n_0 = n_1 m + C_0$$

式中：

$$n_1 = \left[\frac{n_0}{m}\right]$$

显然，有

$$n_1 = C_k m^{k-1} + C_{k-1} m^{k-2} + \cdots + C_1$$

于是，同样的道理，可以得出 C_1 **是** n_1 **除以** m **的余数**，$C_1 = n_1 \bmod m$，**可写为**

$$n_1 = n_2 m + C_1$$

式中：

$$n_2 = \left[\frac{n_1}{m}\right]$$

C_2 **是** n_2 **除以** m **的余数**，$C_2 = n_2 \bmod m$，**可写为**

$$n_2 = n_3 m + C_2$$

式中：

$$n_3 = \left[\frac{n_2}{m}\right]$$

$$\vdots$$

C_k **是** n_k**除以** m **的余数**，$C_k = n_k \bmod m$，**可写为**

$$n_k = n_{k+1} m + C_k$$

式中：

$$n_{k+1} = \left[\frac{n_k}{m}\right]$$

直至 $n_{k+1} = 0$，**运算完成。结论是各商数** n_j **可写为**

$$n_j = \left[\frac{n_{j-1}}{m}\right] \qquad j = 1, 2, \cdots \tag{A-2}$$

各余数可写为

$$C_j = n_j \bmod m \tag{A-3}$$

【例 1】　将 429 **表示为七进制数。**

解：$m = 7$，**由以上定理可得**：

$$n_0 = 429 \qquad C_0 = 429 \bmod 7 = 2$$

$$n_1 = \left[\frac{429}{7}\right] = 61 \qquad C_1 = 61 \bmod 7 = 5$$

$$n_2 = \left[\frac{61}{7}\right] = 8 \qquad C_2 = 8 \bmod 7 = 1$$

$$n_3 = \left[\frac{8}{7}\right] = 1 \qquad C_3 = 1 \bmod 7 = 1$$

$$n_4 = \left[\frac{1}{7}\right] = 0$$

所以

$$429 = 1\times 7^3 + 1\times 7^2 + 5\times 7^1 + 2\times 7^0 = (1152)_7$$

A.1.3　最大公因数算法

定理一：**若** $a = bq + r$ **则**

$$\gcd\{a, b\} = \pm \gcd\{b, r\} \tag{A-4}$$

证明：**设** $d = (a, b)$，$d' = (b, r)$，**则** $d \mid (a - bq)$**即** $d \mid r \Rightarrow d \mid d'$，**可令** $d' = hd$；**另**

一方面，$d' \mid (bq+r)$**即** $d \mid a \Rightarrow d \mid d'$，**可令** $d=kd'$，**联立得**：$d'=hd=hkd'$，**所以** $hk=1$，$h=k=\pm 1$，$d=\pm d'$，**即**$(a, b)=\pm(b, r)$。

欧几里德算法(辗转相除法)：定理一意味着求两数的最大公约数，只需求其中较小者与它们余数的最大公约数即可。由此便得到一种求最大公约数的方法，称为欧几里德算法。

【例 2】 求 $\gcd\{771, 201\}$。

解： $r_1=771 \bmod 201=168$，**根据定理一有**：$\gcd\{771, 201\}=\gcd\{201, 168\}$；

$r_2=201 \bmod 168=33$，**根据定理一有**：$\gcd\{201, 168\}=\gcd\{168, 33\}$；

$r_3=168 \bmod 33=3$，**根据定理一有**：$\gcd\{168, 33\}=\gcd\{33, 3\}$；

$r_4=33 \bmod 3=0$，**至此已除尽。**

所以 $$\gcd\{771, 201\}=3$$

定理二：若 $d=\gcd\{a, b\}$，**则存在整数** p **和** q，**使**

$$pa+qb=d \tag{A-5}$$

只需把辗转相除的过程逆向代回，即可证明此定理。

【例 3】 根据上例，找出$\{771, 201\}=3$ **的** p **和** q。

解：
$$\begin{aligned} 3 &= 168-33\times 5=168-(201-168)\times 5=168\times 6-201\times 5 \\ &= (771-201\times 3)\times 6-201\times 5 \\ &= 771\times 6-201\times 23 \end{aligned}$$

所以 $$p=6, \quad q=-23$$

定理三：若 $a \mid bc$ **且** $\gcd\{a, b\}=1$，**则** $a \mid c$**(证明从略)**。

推论：若素数 $p \mid a_1 a_2 \cdots a_n$，**则必存在** k，$1 \leqslant k \leqslant n$，**使** $p \mid a_k$。

合数分解定理：每一个正合数必能唯一地写成若干正素数的乘积(允许相同因子存在，不考虑因子的顺序)：

$$C=p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n} \tag{A-6}$$

A.1.4 因式分解方法

因式分解(也就包含了证明一个数是否为素数)是密码学中一个重要的问题。这里介绍其中几个算法。

1. 斯泰因(Stein)算法

首先罗列几个有关最大公约数(n_1, n_2)**的性质：**

(1) **若** n_1，n_2 **都是偶数，则**

$$\gcd\{n_1, n_2\}=2\gcd\left\{\frac{n_1}{2}, \frac{n_2}{2}\right\} \tag{A-7}$$

(2) **若** n_1 **为偶数，** n_2 **为奇数，则**

$$\gcd\{n_1, n_2\}=\gcd\left\{\frac{n_1}{2}, n_2\right\} \tag{A-8}$$

(3) **若** $n_1 n_2$ **都是奇数，则**

$$\gcd\{n_1, n_2\}=\gcd\left\{\frac{n_1-n_2}{2}, n_2\right\} \tag{A-9}$$

斯泰因分解法：

S1：$C\leftarrow 0$，$n=n_1\cdot n_2$。

S2：if n_1，n_2 **均偶** then $n_1\leftarrow\frac{n_1}{2}$，$n_2\leftarrow\frac{n_2}{2}$，$C\leftarrow C+1$，**转** S2 else **转** S3。

S3：**若** n_1 **为奇**，n_2 **为偶**，**则** $t\leftarrow n_1$，$n_1\leftarrow n_2$，$n_2\leftarrow t$，**否则转** S4。

S4：**若** n_1 **为偶**，n_2 **为奇**，**则** $n_1\leftarrow\frac{n_1}{2}$，**转** S4，**否则转** S5。

S5：**若** $n_1-n_2<0$，**则** $t\leftarrow n_1$，$n_1\leftarrow n_2$，$n_2\leftarrow t$，**否则转** S6。

S6：$n_1\leftarrow\frac{n_1-n_2}{2}$。

S7：**若** $n_1=0$，**则输出** $g=2^c n_2$，**否则转** S4。

2. 费尔玛(Fermat)算法

n **为正奇数，若能将** n **写成** $n=u^2-v^2$ **的形式，则**

$$n=(u+v)\cdot(u-v)=a\cdot b$$

式中：

$$a=u+v,\ b=u-v$$

据此，费尔玛算法尝试寻找两个平方数。在 a **与** b **大小接近的情况下，因** $u=\frac{1}{2}(a+b)$ **比** $\sqrt{n}$ **稍大，故可以从** $[\sqrt{n}]$ **找起。**

例如，欲分解 54 227，**因为** $[\sqrt{54\ 227}]\approx 232$，**所以测试**

$$v^2=u^2-n=233^2-54\ 227=62$$

显然不是完全平方数；比 233 **大一点再测试**

$$v^2=234^2-54\ 277=529=23^2$$

成功了。于是：

$$54\ 277=234^2-23^2=(234+23)\times(234-23)=257\times 211$$

有时很难找到满足 $n=u^2-v^2$ **的完全平方数，则此时可以先寻找满足** $u^2-v^2=kn$ **的** u **和** v。

【例 4】　分解 $n=24\ 929$。

解：取 $k=3$，**则有**

$$[\sqrt{3\times 24\ 929}]=[\sqrt{74\ 787}]\approx 273$$

此时测试得 $274^2-74\ 787=289=17^2$；**于是：**

$$u+v=274+17=291,\ u-v=274-17=257$$

得到

$$291\times 257=3\times 24\ 929$$

所以

$$24\ 929=97\times 257$$

3. 波拉德(pollard)算法

由费尔玛算法的推广， $u^2-v^2=kn$ **出发，此方程等价于** $u^2=v^2 \bmod n$，**它叫做二次同余式。其普遍形式为**

$$x^2=a^2 \bmod n$$

$x=\pm a$ 是它的平凡解。若还存在一个非平凡解 $x^2=b^2 \bmod n$，但 b 不满足 $x=b \bmod n$，则必有 $0=(a^2-b^2) \bmod n$，即 $n|(a^2-b^2)=(a+b)(a-b)$，于是分解 n 的问题转化为求 $\gcd\{n, a+b\}$ 和 $\gcd\{n, a-b\}$ 的问题。

例如，$x^2=4 \bmod 77$ 的平凡解是 $x=2$，非平凡解是 $x=9$，则因 $\gcd\{77, 9+2\}=11$，$\gcd\{77, 9-2\}=7$，所以 $77=11\times 7$。

一个启发性程序为

S1：$i\leftarrow 1$，产生一个随机数 x，$(0<x_1<n-1)$，$y<x_1$，$k\leftarrow 2$。

S2：若 y 不是 n 的因子，则转 S3。

S3：$i\leftarrow i+1$，$x_i\leftarrow (x_{i-1}^2-1) \bmod n$，$d\leftarrow \gcd\{y-x_i, n\}$。

S4：若 $d\neq 1$，且 $d\neq n$，则 d 是 n 的因子。

S5：若 $i=k$，则 $y\leftarrow x_i$，$k\leftarrow 2k$，转 S2，否则直接转 S2。

A.1.5 模运算

1. 同余

定义：若 $a\equiv b \bmod m$，则称 a 和 b 对模 m 运算同余。

同余意味着：

$$a \bmod m = b \bmod m$$

或写为 $m|(a-b)$（比如日常计时，13 时=1 时，12 为模）。

性质：自反性：$a\equiv a \bmod m$ (A-10)

对称性：若 $b\equiv a \bmod m$，则 $a\equiv b \bmod m$ (A-11)

传递性：若 $a\equiv b \bmod m$，$b\equiv c \bmod m$，则 $a\equiv c \bmod m$ (A-12)

2. 模运算法则

若 $a\equiv b \bmod m$，$c\equiv d \bmod m$，则有：

(1) $a\pm c\equiv b\pm d \bmod m$（加减后求模等于分别求模后加减） (A-13)

(2) $ac=bd \bmod m$ (A-14)

证明：设 $a=(km+b)$，$c=(hm+d)$，则

$$ac=(km+b)(hm+d)=khm^2+kdm+hbm+bd=bd \bmod m$$

(3) 若 $ac=bc \bmod m$ 且 $\gcd\{c, m\}=1$（互素），则 $a=b \bmod m$。

(4) 推广之，若 $ac=bc \bmod m$ 且 $\gcd\{c, m\}=d$，则 $a=b \bmod \left(\frac{m}{d}\right)$。

3. 大数的模幂运算

由乘法模运算法则可推出平方的模运算法则：

若 $a=b \bmod m$，则

$$b^2 \bmod m = a^2 \bmod m = (b \bmod m)^2 \bmod m$$

表明当 b 值较大时，可以先求模再平方。

进而推广到 n 次方时，有

$$b^n \bmod m = (b \bmod m)\cdot(b^{n-1} \bmod m) \bmod m \quad \text{(A-15)}$$

这可以逐次降低幂次。

【例 5】 计算 $675^{12} \bmod 23$。

解：因为

$$675^{12}=(3^3\times5^2)^{12}=(27\times25)^{12}$$

$$27^{12} \bmod 23=4^{12} \bmod 23=64^4 \bmod 23=18^4 \bmod 23$$

$$=(18^2 \bmod 23)^2 \bmod 23=2^2 \bmod 23=4$$

$$25^{12} \bmod 23=2^{12} \bmod 23=(2^6)^2 \bmod 23=18^2 \bmod 23=2$$

所以

$$675^{12} \bmod 23=(4\times2)\bmod 23=8$$

计算机编程计算时可参考下述思路：首先将底数放入累加器中，将指数化成二进制数，从最高位的 1 开始计算。指数每右移一位，累加器中的数值就应当自乘一次(变成原来的平方)，然后根据指数当前位的数值作不同处理：若当前位是 1，则将累加器的值乘以底数，仍放在累加器中；若当前位是 0，则不做什么，直接继续下一轮的右移处理。

用 C 语言编写的求 $s=x^y \bmod n$ 的源程序如下：

```
Unsigned long qe2(unsigned long x, unsigned long y, unsigned long n){
    unsigned long s, t, u;
    int i;
    s=1; t=x; u=y;
    while (u) {
        if (u&1) s=(s*t)% n;
        u>>1;
        t=()% n;
        }
    return(s)
}
```

利用预处理，还可以计算得更快。假定求 $a^e \bmod n$，这里

$$e=26235947428953663183191$$

$$=\underline{1011000}\ \underline{11100}\ \underline{1000000}\ \underline{1110}\ \underline{10010100}\ \underline{1110}\ \underline{101000000}\ \underline{101}$$

$$\underline{111}\ 00\ 0000\ \underline{1111100}\ \underline{1100}\ \underline{1010}\ \underline{10111}$$

下划线表示取计算窗口的大小，现在是 3 位。从左开始取起，遇见 1 才开窗口，跳过中间的 0，总保证窗口第一位是 1，直到最后一组，因它不足 3 位，只好 2 位。

以 $c=a^{\underline{1011000}\ \underline{111}}$ 为例，可写 $c=[(a^{101})^{2^3}(a^{100})]^{2\times2^3}(a^{111})$。3 位二进制数只有 8 个，去掉全 0 的一个，非 0 开头的是 a^{111}、a^{101}、a^{110}、a^{100}、a^{10}、a^{11}、a^{1}，可以预先算出来，用到哪一个就把哪一个做模运算，然后代回继续计算。

4. 模逆运算

定义：若 $a\,b=1 \bmod m$，则称 a 和 b 互为模 m 运算下的逆元，简称模逆元。记作：

$$a=b^{-1} \bmod m \quad \text{或} \quad b=a^{-1} \bmod m \tag{A-16}$$

模逆元存在定理：a 存在模 m 逆元的条件是 a 与 m 互素。

证明：① 必要性：若 a 与 m 不互素，即 $g=(a,m)>1$，则存在 p 和 q，使

$$g=pa+qm=pa \bmod m$$

即 $pa=g\neq1 \bmod m$，表明 a 无模 m 逆元。

② **充分性**：若 a 与 m 互素，即 $\gcd\{a, m\}=1$，则有 $pa+qm=1$，即 $pa=1 \bmod m$，所以 $p=a^{-1}$ 存在。

求模 m 逆元的算法：

由上面的证明过程可见，推出了 $\gcd\{a, m\}=pa+qm=1$，同时也就找到了 $a^{-1}=p$，否则，推导结果若 $pa+qm \neq 1$，则 a 无模 m 逆元。求(a, m)的方法之一就是欧几里德算法。(后面还将介绍另外的方法。)

算法描述如下：

S1：$n_1 \leftarrow m$，$n_2 \leftarrow a$，$q \leftarrow 0$，$p \leftarrow 1$。

S2：求 k、r，使 $n_1=kn_2+r$。

S3：若 $r \neq 0$，则 $n_1 \leftarrow n_2$，$n_2 \leftarrow r$，$t \leftarrow p$，$p \leftarrow q-kp$，$q \leftarrow t$，转 S2，否则转 S4。

S4：若 $n_2 \neq 1$，则给出不存在 a^{-1} 的信息，结束，否则转 S5

S5：若 $p<0$，则 $p \leftarrow q+m$。

S6：输出 $a^{-1}=p$，结束。

A.2 同余类与同余方程

A.2.1 同余类

若 $a \bmod m=b \bmod m$，则 a 与 b 同余。

实际上，模 m 下同余的整数可能有很多个，所有与 r 同余的整数是 $r+km(k=0, \pm 1, \pm 2\cdots)$，它们构成一个同余类，记作$[r]$。同余类也称为剩余类。

(1) 任给一个整数 r，不难找到它在模 m 下的同余类。如 $m=6$，则$[0]=0, 6, 12, 18, \cdots$；$[1]=1, 7, 12, \cdots$；$[2]=2, 8, 14, \cdots$；$[3]=3, 9, 15, \cdots$；$[4]=4, 10, 16, \cdots$；$[5]=5, 11, 17, \cdots$。一般总用 $0, 1, 2, \cdots, m-1$ 中的那个 r 作为该类的代表。

(2) 一组(m 个)整数 $r_1 r_2 \cdots r_m$，如果分属于不同的同余类，则称它们是模 m 的完全剩余类集，记作 Z_m，如$[0, 1, 2, \cdots, m-1]$。

(3) 若 r 与 m 互素，则$[r]$中所有的元素均与 m 互素。

(4) 模 m 的完全剩余类集 Z_m 中，与 m 互素的元素的子集称做既约剩余类，记作 Z_m^*。

A.2.2 欧拉 (Euler) 定理

1. 欧拉数

(1) 定义：与 m 互素的同余类的个数叫做欧拉数，记作 $\phi(m)$。

例如，$m=5$ 时，$\phi(m)=4$，它们是$[1]$，$[2]$，$[3]$，$[4]$；$m=6$ 时，$\phi(m)=2$，它们是$[1]$，$[5]$。

(2) 当 m 为素数时：

$$\phi(m) = m-1 \tag{A-17}$$

(3) 若 m_1 与 m_2 互素，则

$$\phi(m_1 m_2) = \phi(m_1)\phi(m_2) \tag{A-18}$$

例如，$\phi(30)=\phi(5)\phi(6)=4\times2=8$，**它们是**[1]，[7]，[11]，[13]，[17]，[19]，[23]，[29]。

(4) **当** m_1 **和** m_2 **均为素数时，则**

$$\phi(m_1m_2)=(m_1-1)(m_2-1)$$

(5) **若** $m=p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$（$p_i$ **为素因子**，$i=1,2,\cdots,k$），**则**

$$\phi(m)=m\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_k}\right) \tag{A-19}$$

奇怪的是，竟然与 α **无关。**

例如，$30=2\times3\times5$，$\phi(30)=30\times\left(1-\frac{1}{2}\right)\left(1-\frac{1}{3}\right)\left(1-\frac{1}{5}\right)=8$

2. 欧拉(Euler)定理

若 a **与** m **互素，则**

$$a^{\phi(m)}\equiv1 \bmod m \tag{A-20}$$

证明：设 $\phi(m)=k$，**且这** k **个不同的、与** m **互素的同余类的代表元素是** r_1，r_2，…，r_k，**由于** a **与** m **互素，因此** ar_1，ar_2，…，ar_k **这** k **个数也一定与** m **互素，且互不相同。(否则若** $ar_i=ar_j \bmod m$，**将导致** $r_i=r_j \bmod m$，**与假设矛盾。)就是说，** ar_1，ar_2，…，ar_k **仍还是** k **个与** m **互素的、分属于不同的同余类中的另一个元素。从微观上看，类的归属关系可能已经改变，** ar_j **也许会属于原来的**[r_i]**类，但从宏观上讲，共** k **类，每类一个元素，一个也没多，一个也没少，因此它们的乘积在模运算下，是同余的：**

$$ar_1ar_2\cdots ar_k=r_1r_2\cdots r_k \bmod m$$

化简即

$$a^k=1 \bmod m$$

① **推论** 1(**费尔玛**(Fermat)**定理**)：**若** p **是素数，(必然与** a **互素)，因** $\phi(p)=p-1$，**则：**

$$a^{p-1}\equiv1 \bmod p \tag{A-21}$$

② **推论** 2：**若** p **是素数，则**

$$a^p=a \bmod p \tag{A-22}$$

③ **推论** 3：**因为**

$$a^{\phi(m)}=a\cdot a^{\phi(m)-1}\equiv1 \bmod m$$

所以

$$a^{-1}=a^{\phi(m)-1} \bmod m \tag{A-23}$$

这就给出了求乘法逆元的另一方法。这里不要求 m **是素数，只要求** m **与** a **互素即可。**

④ **对于素数** p：**由①便有：**

$$a^{-1}=a^{p-2} \bmod p \tag{A-24}$$

【例 1】 $a=2$，$p=11$，**计算** a **的模** p **逆元。**

解：

$$a^{-1}=2^9 \bmod 11=512 \bmod 11=6$$

【例 2】 **计算** $3^{102} \bmod 11$。

解：因为 $\phi(11)=10$，**所以**

$$3^{10}=1 \bmod 11 \quad (\textbf{欧拉定理})$$

$$\begin{aligned}3^{102}&=3^{100}\times3^2 \bmod 11\\&=\{[(3^{10})^{10} \bmod 11]\times[3^2 \bmod 11]\} \bmod 11\\&=(1^{10}\times9) \bmod 11=9\end{aligned}$$

【例 3】 $a=7$，$m=31$，计算 a 的模 m 逆元。

解：
$$a^{-1}=a^{29} \bmod 31=a^{16}a^{8}a^{4}a^{1} \bmod 31$$

因为
$$7^{2}=18 \bmod 31, 7^{4}=18^{2} \bmod 31=14 \bmod 31$$
$$7^{8}=14^{2} \bmod 31=10 \bmod 31, 7^{16}=10^{2} \bmod 31=7 \bmod 31$$

所以
$$a^{-1}=7\times10\times14\times7 \bmod 31=9 \bmod 31$$

A.2.3 威尔森（Wilson）定理

为了判定一个数是不是素数，Wilson 指出，p 为素数的充要条件是：

$$(p-1)!\equiv-1 \bmod p \quad \text{或者说是} \quad (p-1)!\equiv p-1 \bmod p \tag{A-25}$$

证明：① 必要性（即由 p 是素数推导出 $(p-1)!=-1 \bmod p$）。首先 $p=2$ 时，$(p-1)!=1=-1 \bmod 2$ 成立。因此可假定 p 为大于 2 的素数，即

$$(p-1)!=(p-1)(p-2)\cdots3\cdot2\cdot1$$

由于 p 为素数，$1\leqslant a<p$，a 和 p 一定互素，因此 a^{-1} 一定存在，且一定仍在 $1\leqslant a^{-1}<p$ 范围取值。证明的思路就是从 $aa^{-1}=1 \bmod p$ 出发，将 $(p-1)!$ 中的各个因子都用来寻找自己的模逆元，两两归并为 1，最后看剩下什么。

第一个因子是 $a=1$，显然 $a^{-1}=a=1$，才能使 $aa^{-1}=1 \bmod p$。最后一个因子是 $a=p-1 \bmod p=-1 \bmod p$，也只有 $a^{-1}=a=-1$，才能使 $aa^{-1}=1 \bmod p$。

其他还有 $p-3$（一定是偶数）个因子，它们是 $a=2, 3, \cdots, p-3, p-2$。它们的逆元 a^{-1} 存在，并且仍然是在 $[2, 3, \cdots, p-3, p-2]$ 的范围内取值，而这个范围内的 $a^{-1}\neq a$，因此它们只能是两两配对互逆，使得：

$$2\cdot3\cdot4\cdot5\cdots(p-3)(p-2)=1 \bmod p$$

于是：
$$(p-1)!=1\cdot2\cdot3\cdot4\cdots(p-3)(p-2)(p-1) \bmod p=(p-1) \bmod p$$

或
$$(p-1)! \bmod p=-1 \bmod p$$

比如 $p=7$ 时，aa^{-1} 配对是 $1\leftrightarrow1$，$2\leftrightarrow4$，$3\leftrightarrow5$，$6\leftrightarrow6$。

② 充分性（用反证法）。假定 $(p-1)!=-1 \bmod p$，但 p 却为合数。

设 a 为 p 的一个大于 1 的因子：$a|p$，由 $(p-1)!+1=0 \bmod p$ 知 $p|[(p-1)!+1]$，于是得到 $a|[(p-1)!+1]$。又因为 $(p-1)!$ 中包含了所有小于 p 的因子，必然 $a|(p-1)!$。二者都成立则导致 $a|1$，这与 $a>1$ 相矛盾（与 p 为合数的假设相矛盾），因此结论只能是 p 为素数。

A.2.4 线性同余方程

1. 定义

若 x_1 满足线性方程：

$$ax=b \bmod m \tag{A-26}$$

容易想到 x_1 的同余类 $x_2=x_1 \bmod m$ 亦满足这个线性方程。推而广之，一切模 m 下同余类均满足这个线性同余方程。如 4，9，14，19……都在模 5 运算下余 4，它们均满足方程：

$$2x=3 \bmod 5 \quad \text{或} \quad 3x=2 \bmod 5 \quad \text{或} \quad 4x=1 \bmod 5\cdots$$

同余类所满足的模 m 下的一次方程 $ax=b \bmod m$ 叫做线性同余方程。

然而，并非任何线性同余方程 $ax=b \bmod m$ 都有解，只有 a 与 b 满足一定条件，构成的同余方程才有解。

2. 线性同余方程有解定理

$ax=b \bmod m$ 有解的充要条件是 $d \mid b$，这里 $d=(a, m)$。也就是说，同余方程要求 a 与 m 的最大公因数能除尽 b。

证明：① 必要性。若有解，设解为 x_0，且满足 $ax_0=b \bmod m$，即

$$ax_0 - km = b$$

因 $d=(a, m)$，必有 $d \mid (ax_0-km)=b$。

② 充分性。若 $d \mid b$，可设 $b=b'd$，因 $d=(a, m)$，可设

$$a = a'd,\ m = m'd$$

显然 a' 与 m' 互素，一定存在 pq 使 $pa'+qm'=1$，于是：

$$b = pba' + qbm' = b'pda' + b'qdm' = pb'a + qb'm$$

表明 $a(pb')=b \bmod m$，pb' 就是原方程的解。

特例：当 $d=(a, m)=1$（互素）时，$ax=b \bmod m$ 一定有解。

3. 求解线性同余方程的方法

（1）在找到了一个解 x_0 满足 $ax_0=b \bmod m$ 的前提下，一切满足 $x=x_0 \bmod \left(\frac{m}{d}\right)$ 的 x 都是原方程的解。这里 $d=(a, m)$，可令

$$m' = \frac{m}{d},\ a' = \frac{a}{d}$$

于是：

$$ax = a(x_0 + km') = ax_0 + akm' = ax_0 + a'dkm' = ax_0 + a'km = ax_0 \bmod m$$

【例 4】 已知 $x_0=4$ 是同余方程 $2x=2 \bmod 6$ 的一个解，求此方程的解。

解： 因为　$d=(2, 6)=2,\ m'=\frac{m}{d}=3$

所以　$x=x_0 \bmod m'=4 \bmod 3$

表达了一切满足原方程的解。如 $x=1, 4, 7, 10, 13, 16, \cdots$

（2）利用模逆元直接求解同余方程。由 $ax=b \bmod m$，就有

$$x = a^{-1}b \bmod m$$

【例 5】 已知 $7x=22 \bmod 31$，$a=7$，$m=31$，求解 x。

解： 因为 $7\times 9=1 \bmod 31$，所以

$$7^{-1} = 9 \bmod 31$$

所以有　$x=9\times 22 \bmod 31=12 \bmod 31$

A.2.5　线性同余方程组——中国剩余定理

若

$$\begin{cases} x=b_1 \bmod m_1 \\ x=b_2 \bmod m_2 \end{cases}$$

有一个共同解 x_1，则

$$x_1 = b_1 + k_1m_1 = b_2 + k_2m_2$$

可见 $$k_2 m_2 = (b_1 - b_2) \bmod m_1$$

而 k_2 的同余方程有解的充分条件是$(m_1, m_2) | (b_1 - b_2)$。推而广之，就得中国剩余定理。

中国剩余定理：设 $m_1, m_2, \cdots, m_k$ 是 k 个两两互素的正整数，则联立方程组：

$$x = b_i \bmod m_i \qquad (i = 1, 2, \cdots, k) \tag{A-27}$$

在最小公倍数$[m_1 m_2 \cdots m_k]$内有唯一解。或者说，联立方程在最小公倍数$[m_1 m_2 \cdots m_k]$为模的运算下有唯一解。

令 $$M = m_1 \cdot m_2 \cdots m_k,\ M_j = \frac{M}{m_j} \qquad (j = 1, 2, \cdots, k)$$

由于 $m_1 m_2 \cdots m_k$ 两两互素，因此$(M_j, m_j) = 1$(互素)，逆元存在。求出的模逆元 y_j 为

$$y_j = M_j^{-1} \bmod m_j \qquad (j = 1, 2, \cdots, k)$$

下面证明方程组的解为

$$x = b_1 M_1 y_1 + b_2 M_2 y_2 + \cdots + b_k M_k y_k \tag{A-28}$$

实际上当 $j \neq h$ 时，$m_h | M_j$，故 $M_j = 0 \bmod m_h$，所以

$$x \bmod m_h = b_h M_h y_h \bmod m_h = b_h \bmod m_h$$

只剩下这一项不为零。此即联立方程组中的第 h 个子方程。

当 h 取遍 k，就证明了 x 满足所有子方程。

此类问题最早记载于中国古代孙子算法，俗称韩信点兵：16 一数三三数余 A，五五数余 B，七七数余 C，求该数”。答案是四句诗：“三人同行古来稀，五朵梅花二十一，七子相逢正月半，减百零五便得知”。其意思是方程组：

$$\begin{cases} x = A \bmod 3 \\ x = B \bmod 5 \\ x = C \bmod 7 \end{cases}$$

的解为 $$x = 70A + 21B + 15C \bmod 105$$

【例 6】 求解同余方程组

$$\begin{cases} x = 1 \bmod 3 & (1) \\ x = 2 \bmod 5 & (2) \\ x = 3 \bmod 7 & (3) \end{cases}$$

解：解法一(算术法)。

$$M = 3 \times 5 \times 7 = 105$$

$$M_1 = 35,\ M_2 = 21,\ M_3 = 15$$

$$y_1 = 2,\ y_2 = 1,\ y_3 = 1$$

所以 $$x = 1 \times 35 \times 2 + 2 \times 21 \times 1 + 3 \times 15 \times 1 \bmod 105 = 167 \bmod 105 = 52$$

解法二(代数法)。由(1)知 $x = 3u + 1$，代入(2)得

$$3u + 1 = 2 \bmod 5$$

即 $3u = 1 \bmod 5$，容易看出 $u_0 = 2$ 是一个解。且因 $d = (3, 5) = 1$，$m' = \frac{m}{d} = 5$，于是一切 $u = 2 \bmod 5$ 都是(2)的解。

由此可写 $u = 5v + 2$，代回得

$$x = 3u + 1 = 15v + 7$$

再代入(3)得

$$15v + 7 = 3 \bmod 7$$

即

$$15v = -4 \bmod 7 = 3 \bmod 7$$

容易看出 $v_0 = 3$ 是一个解。且因 $d = (15, 7) = 1$，一切 $v = 3 \bmod 7$ 都是(3)的解。此即 $v = 7w + 3$，再代回得

$$x = 15v + 7 = 15(7w + 3) + 7 = 105w + 52$$

所以

$$x = 52 \bmod 105$$

A.2.6　平方剩余(二次剩余)

考察下列同余方程：

$$x^2 = 1 \bmod 5,\ x^2 = 2 \bmod 5,\ x^2 = 3 \bmod 5,\ x^2 = 4 \bmod 5$$

不难发现，$x = 1$ 和 $x = 4$ 满足方程 $x^2 = 1 \bmod 5$；$x = 2$ 和 $x = 3$ 满足方程 $x^2 = 4 \bmod 5$；而方程 $x^2 = 2 \bmod 5$ 和 $x^2 = 3 \bmod 5$ 都没有整数解。

再考察同余方程 $x^2 = b \bmod 7 (b = 1, 2, 3, 4, 5, 6)$，同样发现 $b = 1, 2, 4$ 时方程有解，$b = 3, 5, 6$ 时方程无解。

1. 定义

若 a 与 p 互素，p 为奇素数(即 $p \neq 2$)，则方程：

$$x^2 = a \bmod p \tag{A-29}$$

有整数解的那些 a 值称为模 p 的平方剩余，记作 $a \in \mathrm{QR}$；无整数解的那些 a 值称为模 p 的非平方剩余，记作 $a \in \mathrm{NQR}$。

2. 勒让德(Legendre)符号

定义：

$$L(a, p) = \begin{cases} 0 & a \text{ 整除 } p \\ 1 & a \text{ 为模 } p \text{ 的平方剩余} \\ -1 & a \text{ 为模 } p \text{ 的非平方剩余} \end{cases}$$

进而，因 p 为奇素数，a 必不能整除 p，故可排除第一种情况，从而有

$$L(a, b) \equiv \left(\frac{a}{p}\right) = \begin{cases} 1 & a \text{ 为模 } p \text{ 的平方剩余} \\ -1 & a \text{ 为模 } p \text{ 的非平方剩余} \end{cases}$$

例如，$p = 7$ 时有：

$$\begin{cases} \left(\frac{1}{7}\right) = \left(\frac{2}{7}\right) = \left(\frac{4}{7}\right) = 1 \\ \left(\frac{3}{7}\right) = \left(\frac{5}{7}\right) = \left(\frac{6}{7}\right) = -1 \end{cases}$$

3. 定理

若 p 为奇素数，则：

$$\left(\frac{a}{p}\right) \equiv L(a, p) = a^{\frac{p-1}{2}} \bmod p \tag{A-30}$$

此定理给出了计算勒让德符号的方法，并由此判定 a 是否为平方剩余。

证明：(1) **若** a **为模** p **的平方剩余，即** $x^2=a \bmod p$ **有解，设解为** x'，**则利用费尔玛**(Fermat)**定理**：

$$a^{(p-1)/2}=((x')^2)^{(p-1)/2}=(x')^{p-1}=1 \bmod p$$

有：

$$a^{(p-1)/2} \bmod p=1$$

(2) **若** a **为模** p **的非平方剩余，即** $x^2=a \bmod p$ **无解，则对于每一个** $1<i<p-1$ **的整数** i，**总对应一个** j，**使得** $i \cdot j=a \bmod p$ **成立，且** $i \neq j$，**于是** 1，2，…，$p-1$ **可分为** $(p-1)/2$ **对，每一对的乘积为** a，**因此**：

$$(p-1)!=a^{(p-1)/2} \bmod p$$

而威尔森(Wilson)**定理告诉我们**：

$$(p-1)!=-1 \bmod p$$

因而有：

$$a^{(p-1)/2} \bmod p=-1$$

合并则是：

$$a^{(p-1)/2} \bmod p=\left(\frac{a}{p}\right)$$

例如，$p=7$，**不难验证**，$a=1$，2，4 **时**$\left(\frac{a}{p}\right)=a^3 \bmod p=1$；**而** $a=3$，5，6 **时**$\left(\frac{a}{p}\right)=a^3 \bmod p=-1$。

【例 7】 判定 5 **是否为** 23 **的平方剩余**。

解：$p=23$，$\frac{p-1}{2}=11$，$L(5,23)=5^{11} \bmod 23$

因为 $11=8+2+1$，**而**

$$5^2=2 \bmod 23,\ 5^4=2^2 \bmod 23=4,\ 5^8=4^2 \bmod 23=16$$

所以

$$5^{11}=5^8 \times 5^2 \times 5^1 \bmod 23=16 \times 2 \times 5 \bmod 23=-1 \bmod 23$$

可见 5 **不是** 23 **的平方剩余**。(**即** $x^2=5 \bmod 23$ **无解**。)

4. 性质

(1) **若** p **是奇素数，则** $a=1,2,3,\cdots,p-1$ **中正好有**$\frac{1}{2}(p-1)$**个平方剩余**，$\frac{1}{2}(p-1)$**个非平方剩余**。**实际上，由欧拉定理** $a^{p-1}=1 \bmod p$ **知** $a^{\frac{p-1}{2}}=\pm 1 \bmod p$，**取正** 1 **和负** 1 **的各有** $\frac{p-1}{2}$ **个，分别对应** QR **和** NQR。

(2) **二次剩余方程在有解的情况下，若一个解是** x_0，**则另一个解必为** $p-x_0$。**这是因为若** $x_0^2=a \bmod p$，**则**

$$(p-x_0)^2=(p^2-2px_0+x_0^2) \bmod p=x_0^2 \bmod p$$

二者满足同一个平方剩余方程。例如， $x^2=2 \bmod 7$ **的解为** $x=3$ **和** $x=4$，**满足** $3+4=7$。

5. 运算法则

若 $a=b \bmod p$，**则**：

(1) $$L(a,p)=L(b,p) \tag{A-31}$$

(2) $$L(a,p) \cdot L(b,p)=L(ab,p) \tag{A-32}$$

(3) $$L(a^2,p)=1 \tag{A-33}$$

A.2.7　雅可比(Jacobi)符号

1. 定义

将勒让德符号的素数 p 推广到合数 n：

$$n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$$

可定义雅可比符号：

$$J(a, n) \equiv J\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \cdots \left(\frac{a}{p_k}\right)^{\alpha_k} \tag{A-34}$$

而勒让德符号是雅可比符号中 n 取素数的特例。

当 $p_1 p_2 \cdots p_k$ 为 n 的素因子时，由定义式不难得到 $J(a, n) \equiv \pm 1$，不过它不代表 $x^2 = a \bmod n$ 是否有解。

例如， $J\left(\frac{2}{15}\right) = \left(\frac{2}{3}\right)\left(\frac{2}{5}\right) = (-1)(-1) = 1$，**但 $x^2 = 2 \bmod 3$ 和 $x^2 = 2 \bmod 5$ 均无解。**

又如， $J\left(\frac{1}{15}\right) = \left(\frac{1}{3}\right)\left(\frac{1}{5}\right) = 1$，**但因 $x^2 = 1 \bmod 3$ 和 $x^2 = 1 \bmod 5$ 均有两个解，致使 $x^2 = 1 \bmod 15$ 有四个解，分别为** $x_1 = 1$，$x_2 = 4$，$x_3 = 11$，$x_4 = 14$。

2. 性质

(1)
$$J\left(\frac{1}{n}\right) = 1,\ J\left(\frac{-1}{n}\right) = (-1)^{\frac{n-1}{2}} \tag{A-35}$$

(2)
$$J\left(\frac{ab}{n}\right) = J\left(\frac{a}{n}\right) \cdot J\left(\frac{b}{n}\right) \tag{A-36}$$

(3)
$$J\left(\frac{2}{n}\right) = (-1)^{\frac{n^2-1}{8}} \tag{A-37}$$

即：若 $n = 1 \bmod 8$ 或 $n = 7 \bmod 8$，则 $J\left(\frac{2}{n}\right) = 1$；**若 $n = 3 \bmod 8$ 或 $n = 5 \bmod 8$，则** $J\left(\frac{2}{n}\right) = -1$。

(4)
$$J\left(\frac{a}{b}\right) = J\left(\frac{b \bmod a}{a}\right) \tag{A-38}$$

(5) **a，b 为正奇数，若** $(a, b) \equiv 1$，**则**

$$J\left(\frac{a}{b}\right) J\left(\frac{b}{a}\right) = (-1)^{\frac{(a-1)(b-1)}{4}} \tag{A-39}$$

3. 特例

一个重要的特例是合数 $n = pq$，其中，p，q 均为素数。

(1) **a 为 n 的二次剩余，当且仅当 a 为模 p 的二次剩余，且 a 为模 q 的二次剩余。**

证明：因为　　$J\left(\frac{a}{n}\right) = J\left(\frac{a}{p}\right) \cdot J\left(\frac{a}{q}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$

所以， $J\left(\frac{a}{n}\right) = 1$ **包括：**

$$\left(\frac{a}{p}\right) = 1,\ \left(\frac{a}{q}\right) = 1 \quad \text{和} \quad \left(\frac{a}{p}\right) = -1,\ \left(\frac{a}{q}\right) = -1$$

$J\left(\frac{a}{n}\right)=-1$ 包括：

$$\left(\frac{a}{p}\right)=1,\ \left(\frac{a}{q}\right)=-1 \quad \text{和} \quad \left(\frac{a}{p}\right)=-1,\ \left(\frac{a}{q}\right)=1$$

上面四种情况中，只有$\left(\frac{a}{p}\right)=1$ 且$\left(\frac{a}{q}\right)=1$ 的情况，即

$$\begin{cases} x^2 = a \bmod p \\ x^2 = a \bmod q \end{cases} \tag{A-40}$$

都有整数解时，$x^2=a \bmod n$ 才有整数解，而$\left(\frac{a}{p}\right)$和$\left(\frac{a}{q}\right)$只要有一个为-1 就不行，可见 $J\left(\frac{a}{n}\right)=1$ 不是平方剩余的充分条件。

(2) 由于方程 $x^2=a \bmod p$ 存在$(p-1)/2$ 个二次剩余，方程 $x^2=a \bmod q$ 存在$(q-1)/2$ 个二次剩余，因此方程 $x^2=a \bmod n$ 存在$(p-1)(q-1)/4$ 个二次剩余(即能使方程有解的 a 值)。

(3) 有解的情况下，若 $x^2=a \bmod p$ 的解是 x_1 和 $p-x_1$，$x^2=a \bmod q$ 的解是 x_2 和 $q-x_2$，则 $x^2=a \bmod n$ 的解等价于方程组(A-40)的解。应利用中国剩余定理，由以下 4 个方程组分别求得 4 个解：

$$\begin{cases} x = x_1 \bmod p \\ x = x_2 \bmod q \end{cases}, \quad \begin{cases} x = x_1 \bmod p \\ x = (q-x_2) \bmod q \end{cases}$$

$$\begin{cases} x = (p-x_1) \bmod p \\ x = x_2 \bmod q \end{cases}, \quad \begin{cases} x = (p-x_1) \bmod p \\ x = (q-x_2) \bmod q \end{cases}$$

(4) 特别是，当 p 和 q 都是模 4 余 3 的素数(称做 Blum 数)时：

$$p = 3 \bmod 4 \Rightarrow p+1 = 4k \Rightarrow \frac{1}{4}(p+1) \text{ 为整数}$$

利用

$$\left(\frac{b}{p}\right)=b^{\frac{p-1}{2}} \bmod p=1 \bmod p$$

就有

$$(b^{\frac{p+1}{4}})^2=b^{\frac{p+1}{2}}=b^{\frac{p-1}{2}} \cdot b \bmod p=b \bmod p$$

表明 $x_1=b^{\frac{p+1}{4}}$ 是 $x^2=b \bmod p$ 的一个解，自然 $p-x_1$ 是另一个解。

同理，有 $x_2=b^{\frac{q+1}{4}}$ 和 $q-x_2$ 是 $x^2=b \bmod q$ 的两个解。于是 $x^2=b \bmod (pq)$ 的解由上述两解完全确定。

(5) 当 n 为 Blum 数时，有以下性质：

①

$$J\left(\frac{-x}{n}\right)=J\left(\frac{x}{n}\right) \tag{A-41}$$

② 若 x，$y \in Z_n^*$，满足 $x^2=y^2 \bmod n$，且 $x \neq y$(或 $x \neq -y$) $\bmod n$，则

$$J\left(\frac{x}{n}\right)=-J\left(\frac{y}{n}\right) \tag{A-42}$$

【例 8】 $x^2=b \bmod n$，其中 $p=3$，$q=5$，$n=15$。分析 b 取不同值时，方程的整数解。

解：$p=3$，$q=5$，这时它共有$\frac{1}{4}(p-1)(q-1)=2$ 个 QR，分别是 $b=1$ 和 $b=4$。这是

因为，对于 $b=1$ 有：

$$\left(\frac{b}{p}\right)=b^{\frac{p-1}{2}}=b^{1}=1 \bmod 5 \quad \text{和} \quad \left(\frac{b}{q}\right)=b^{\frac{q-1}{2}}=b^{2}=1 \bmod 5$$

对于 $b=4$ 亦有：

$$\left(\frac{b}{p}\right)=b^{\frac{p-1}{2}}=4^{1} \bmod 3=1 \quad \text{和} \quad \left(\frac{b}{q}\right)=b^{\frac{q-1}{2}}=4^{2} \bmod 5=1$$

不难验证 $x^2=1 \bmod p$ 的解是 $x=1$ 和 $x=p-1=2$；$x^2=1 \bmod q$ 的解是 $x=1$ 和 $x=q-1=4$。两个解集分别为$\{1, 2, 4, 5, 7, 8, 10, 11, 13, 14, \cdots\}$和$\{1, 4, 6, 9, 11, 14, \cdots\}$，所以在$[1, 15-1]$区间的共同解为 $x=1$，$x=4$，$x=11$ 和 $x=14$，且满足互补关系 $1+14=15$ 和 $4+11=15$。

同样，不难验证 $x^2=4 \bmod p$ 的解是 $x=2$ 和 $x=p-2=1$；$x^2=4 \bmod q$ 的解是 $x=3$ 和 $x=q-3=2$。两个解集分别为$\{1, 2, 4, 5, 7, 8, 10, 11, 13, 14, \cdots\}$和$\{2, 3, 7, 8, 12, 13, \cdots\}$，所以在$[1, 15-1]$区间的共同解为 $x=2$，$x=7$，以及另两个解 $15-2=13$，$15-7=8$。

实际上，对此简单问题，我们可以列出全部 $x^2 \bmod n$ 的数值，如表 A.1 所示。

表 A.1　x^2 mod 15 的全部同余类

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14
b	1	4	9	1	10	6	4	4	6	10	1	9	4	1

从表 A.1 中可以看出，只有 $b=1$ 和 $b=4$ 各有 4 个解，分别为 $x=1$，4，11，14 和 $x=2$，7，8，13，它们属于 QR，其他 6 个属于 NQR。

A.3　群　和　域

A.3.1　群(Group)

1. 定义

群(G)是一些元素的集合，元素间定义了一种“乘”$(*)$运算，并满足以下四个条件：

(1) 封闭性：若 a，$b \in G$，则 $c=a*b$ 必$\in G$。

(2) 结合律：若 a，b，$c \in G$，则$(a*b)*c=a*(b*c)$。

(3) 存在恒元：恒元 $e \in G$，对任何 $a \in G$，恒有 $a*e=e*a=a$。

(4) 存在逆元：对任何 $a \in G$，恒有 $b \in G$，使 $a*b=b*a=e$，b 称为 a 的逆元，记作 $b=a^{-1}$。

群所包含的元素(称群元)的个数叫做群的阶。一个集合能否构成群，不在于阶有多大，关键的是满足四个条件。例如，$\{1, -1, i, -i\}$对普通乘法运算构成一个四元群；$\{0, 1\}$ 对模二加法构成二元群；全体整数对普通乘法运算并不能构成群，因为除了 1 和 -1，其他整数在集合中都找不到逆元。

群是为描述某些客观对象而建立的数学模型，常用来描述具有某种对称性的物理系

统，如晶体结构、基本粒子，也常用来描述一组数学变换或操作，如旋转、反射等。

一般情况下，群不一定满足乘法交换律。但满足交换律 $a * b = b * a$ 的群叫做阿贝尔群。

2. 同构

同构是群的一个重要概念。如果两个群中各元素一一对应，而且相应元素的运算结果也一一对应，则两个群同构。同构的群具有相同的结构和性质，可以互相表示。

例如，三元循环移位群 P_3 与三度定轴旋转群 C_3 同构：

(1) P_3 群的三个群元是恒元 P_E、循环左移 P_A 和循环右移 P_B，它们分别为

恒元 P_E：$P_E(1, 2, 3) = (1, 2, 3)$

循环左移 P_A：$P_A(1, 2, 3) = (2, 3, 1)$

循环右移 P_B：$P_B(1, 2, 3) = (3, 1, 2)$

因为 $$P_A[P_A(1, 2, 3)] = P_A(2, 3, 1) = (3, 1, 2)$$

P_A 相继两次移位的效果与 P_B 的效果相同，所以 $P_A * P_A = P_B$。这里，$*$ 运算指相继操作。

同理可知，$P_A * P_B = P_B * P_A = P_E$，表明 P_A 与 P_B 互为逆元，同时也表明 P_3 群是阿贝尔群。

(2) C_3 群的三个群元是恒元 $C_E = C_A C_B$、顺时针三度转 C_A 和逆时针三度转 C_B，见图 A.1。

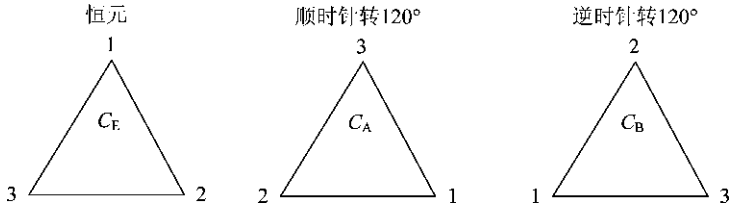

图 A.1 C_3 群的三个群元

将三群元的各种 $*$ 运算的结果列表显示(表 A.2 和表 A.3)，不难体会其封闭性。比较两表，还可以看出，二群不仅各元素一一对应，即 P_E—C_E，P_A—C_A，P_B—C_B，而且 C_3 群元各元素运算结果也与 P_3 群元各相应元素运算结果一致。因此两群同构，记作 $P_3 \cong C_3$。

表 A.2 P_3 群元运算关系

	P_E	P_A	P_B
P_E	E	P_A	P_B
P_A	P_A	P_B	P_E
P_B	P_B	P_E	P_A

表 A.3 C_3 群元运算关系

	C_E	C_A	C_B
C_E	C_E	C_A	C_B
C_A	C_A	C_B	C_E
C_B	C_B	C_E	C_A

3. 循环群

由一个单独元素的一切幂次所构成的群叫做循环群。这个单独元素叫做循环群的生成元。

设某 n 阶循环群由生成元 α 自乘产生：$\{\alpha^0 = 1, \alpha^1, \alpha^2, \cdots, \alpha^{n-1}, \alpha^n = 1\}$。$\alpha^n = 1$ 表明 α 自乘 n 次又回到群的恒元 1，因此我们说 α 的循环级为 n。又因为此循环群的阶数也是 n，所以 α 叫做单位原根，或称本原元素。实际上用除了恒元 1 之外的任何一个群元 α^i(i 任意)

自乘也能生成循环群，但 α^i 的循环级数可能会小于 n，这样的 α^i 就不是单位原根。

例如，$x^3=1$ 有三个根：$\alpha=e^{j2\pi/3}$、$\alpha^2=e^{j4\pi/3}$ 和 $\alpha^3=e^{j6\pi/3}=1$，构成三阶循环群。α 和 α^2 都是 3 级单位原根，$\alpha^3=1$ 是恒元。

$x^4=1$ 有四个根：$\alpha=e^{j2\pi/4}=j$，$\alpha^2=e^{j4\pi/4}=-1$，$\alpha^3=e^{j6\pi/4}=-j$，$\alpha^4=e^{j8\pi/4}=1$，构成四阶循环群。α 和 α^3 是 4 级单位原根，α^2 为 2 级元素，不是单位原根，$\alpha^4=1$ 是恒元。

$x^n=1$ 有 n 个根：$\alpha=e^{j2\pi/n}$，$\alpha^2=e^{j4\pi/n}$，…，$\alpha^n=e^{j2n\pi/n}=1$，它们等分单位圆，构成 n 阶循环群。如果 n 为素数，则除恒元外，其他元素必定都是 n 级单位原根。但若 n 非素数，比如 $n=15=3\times5$，则 α^3 五等分单位圆，自乘 5 次就可得到 1，为 5 级元素。同理，α^3、α^6、α^9、α^{12} 都是 5 级元素。α^5 三等份单位圆，自乘 3 次就可得到 1，为 3 级元素，同理，α^{10} 也是 3 级元素。而 α、α^2、α^4、α^7、α^8、α^{11}、α^{13}、α^{14} 都需要自乘 15 次才能得到恒元，其级数都是 15，它们都是单位原根。

定理：n 阶循环群中有 $\phi(n)$ 个单位原根。$\phi(n)$ 叫做欧拉数，它代表 n 的既约同余类的个数（除以 n 后余数相同的整数为一个同余类，与 n 互素的同余类叫既约同余类）。

例如，$\phi(3)=2$，$\phi(5)=4$，$\phi(15)=8$。

如果不是单位原根，同一级的群元再包括恒元，总可以构成原来循环群的一个子群。所谓子群，是由母群的部分元素构成的群，循环群的子群仍然是循环群。

A.3.2　域（Field）

1. 定义

域也是一种集合 F，它至少含有 0 和 1 两个元素，约定“加”（+）和“乘”（*）两种运算，并满足以下三个条件：

（1）F 的元素关于 + 运算构成阿贝尔群，其恒元为 0。

（2）F 中除 0 之外的元素关于 * 运算也构成阿贝尔群，其恒元为 1；$F\backslash\{0\}$ 表示除 0 之外的元素。

（3）对于 a，b，$c\in F$，分配率成立：

$$(a+b)*c=a*c+b*c,\quad c*(a+b)=c*a+c*b$$

实数的集合 R 构成实数域，复数的集合 C 构成复数域，它们的元素无穷多。整数的集合 Z 不能构成域，因为除了 1 和 −1，其他元素没有乘法逆元。

2. 伽罗瓦（Galois）域

有限个元素构成的域叫有限域，又叫伽罗瓦域。典型的例子是素数 p 的完全剩余类集合 Z_p，它所约定的 + 运算和 * 运算是以 p 为模的加和乘，它的元素是 $\{0,1,2,\cdots,p-1\}$。在以 p 为模的运算下，$\{0,1,2,\cdots,p-1\}$ 中的每个数，实际代表余数相同的一类整数，叫做剩余类。如果 n 不是素数，根据模逆元存在定理，剩余类集合 Z_n 中与 n 不互素的那些元素就不存在乘法模逆元，就不能构成域。只有素数 p 的完全剩余类集合 Z_p 才能构成 p 阶有限域，记作 GF(p)。p 阶指域所包含元素的个数为 p。当然，合数 n 的既约剩余类子集 Z_n^* 也能构成有限域，其阶为 $\phi(n)$。

GF(2)域是最常用的伽罗瓦域，它只有 0 和 1 两个元素，在模 2 加和模 2 乘下满足封闭性，如表 A.4 所示。

表 A.4 GF(2)域的模 2 加和模 2 乘

⊕	0	1
0	0	1
1	1	0

⊙	0	1
0	0	0
1	0	1

3. GF(p)域的本原根

以 $p=7$ 为例，我们发现 3 作为一个域元素，它的各次幂在模 7 之下会取遍 GF(7)域的所有非零元素：$3^0 \bmod 7=1$，$3^1 \bmod 7=3$，$3^2 \bmod 7=2$，$3^3 \bmod 7=6$，$3^4 \bmod 7=4$，$3^5 \bmod 7=5$；以后又会重复：$3^6 \bmod 7=1$，$3^7 \bmod 7=7$，$3^8 \bmod 7=2$，… 表明 GF(7)域是由 0 和($3^0 \sim 3^5$)这 6 个元素的循环群构成的，而 3 就是它的本原根。不难验证，5 是 GF(7)域的另一个本原根：$5^0 \bmod 7=1$，$5^1 \bmod 7=5$，$5^2 \bmod 7=4$，$5^3 \bmod 7=6$，$5^4 \bmod 7=2$，$5^5 \bmod 7=3$，并且 GF(7)域只有这两个本原根。

如果观察模 7 之下 2 的各次幂，则发现它们不会取遍所有的域元素，而只是在 3 个元素之间循环：$2^0 \bmod 7=1$，$2^1 \bmod 7=2$，$2^2 \bmod 7=4$，$2^3 \bmod 7=1$，$2^4 \bmod 7=2$，$2^5 \bmod 7=4$ … 因此 2 不是本原根，叫做 GF(7)域的循环阶为 3 的非本原根。同理，不难发现 4 也是 GF(7)域的循环阶为 3 的非本原根，而 6 是 GF(7)域的循环阶为 2 的非本原根。

理论已证明，存在本原根与非本原根并不是 GF(7)域特有的现象，任何 GF(p)域都有 $\phi(p-1)$个本原根，其他域元素则是非本原根。由本原根的各次幂可生成 $p-1$ 阶循环群，而非本原根的各次幂只能生成以($p-1$)的因数为阶的循环群。

比如 GF(13)域有 $\phi(12)=4$ 个本原根，分别为 2、6、7、11，它们的各次幂可生成 12 阶循环群。GF(13)域还有 8 个非本原根，分别是循环阶为 6 的非本原根 4 与 10，循环阶为 4 的非本原根 5 与 8，循环阶为 3 的非本原根 3 与 9，循环阶为 2 的非本原根 12，循环阶为 1 的非本原根 1，即乘法恒元。

本原根有两条重要性质，即若 g 是 GF(p)域的一个本原根，n，k，m 为整数，则：

(1) 能使 $g^n=1 \bmod p$ 成立的充分必要条件是 $n=0 \bmod (p-1)$。

(2) 能使 $g^k=g^m \bmod p$ 成立的充分必要条件是 $k=m \bmod (p-1)$。

A.3.3 GF(p^m)域

任意 q 个元素未必能构成有限域，因为它们不一定能满足封闭性。但若 $q=p^m$，p 为素数，m 为正整数，则 $q=p^m$ 个元素一定能构成有限域 GF(p^m)。

特别是当 $p=2$ 时，得到的有限域 GF(2^m)是 $q(=2^m)$进制符号的集合，共 2^m 个元素。比如 GF(2^3)域包含 8 个元素，GF(2^8)域包含 256 个元素。当然，每个 $q=2^m$ 进制符号都可用长为 m 的二元序列代替，称为 GF(2^m)域的 m 维矢量表示。不过千万注意，这些二元序列不是二进制自然数！元素之间的运算关系不服从自然数运算法则！比如 GF(2^3)域的 8 个元素，由按位模 2 加运算给出元素间的运算关系是：011⊕110=101，显然不是 3+6=9；同理，011⊙110=001，也不是 3×6=18。

显然，$q=2^m$ 不一定是素数，我们这里讨论的 GF(2^m)域，不是指素数 p 的同余类构成 p 阶有限域 GF(p)，而是由一个加运算的恒元 0 与一个 2^m-1 阶循环群构成的有限域，即 $\{0, \alpha^0=1, \alpha^1, \alpha^2, \cdots, \alpha^{q-2}, \alpha^{q-1}=1\}$，连同 0 元素，$q=2^m$ 个域元素应满足方程：

$$x \cdot \prod_{i=0}^{q-2}(x-\alpha^i)=0 \tag{A-43}$$

1. 共轭类

我们可以把 GF(2^m)域中除 0 以外的 $q-1$ 个元素按以下方式分成若干类：设 β 是域中任意非零元素，按 β^j 幂次分类，凡属于 $i=2^k(k=0,1,2,\cdots)$ 的元素都称为 β 的共轭类。如 GF(2^4)域，共 15 个非零元素，取 $\beta=\alpha$，则 α、α^2、α^4、α^8 属于 $\beta=\alpha$ 的共轭类；取 $\beta=\alpha^3$，则 α^3、$(\alpha^3)^2=\alpha^6$、$(\alpha^3)^4=\alpha^{12}$、$(\alpha^3)^8=\alpha^{24}=\alpha^{24-15}=\alpha^9$ 属于 $\beta=\alpha^3$ 的共轭类；取 $\beta=\alpha^5$，则 α^5、$(\alpha^5)^2=\alpha^{10}$ 属于 $\beta=\alpha^5$ 的共轭类；取 $\beta=\alpha^7$，则 α^7、$(\alpha^7)^2=\alpha^{14}$、$(\alpha^7)^4=\alpha^{28}=\alpha^{28-15}=\alpha^{13}$、$(\alpha^7)^8=\alpha^{56}=\alpha^{56-15-15-15}=\alpha^{11}$ 都属于 $\beta=\alpha^7$ 的共轭类；$\beta=\alpha^0=1$ 自成一类。有限群各共轭类的元素都是有限个，各类元素互不重叠，并恰好取尽域中全部非零元素。

属于同一个共轭类的元素，循环级相同，类中元素数目为

$$2^m=1 \bmod n \tag{A-44}$$

式中，n 是该类元素的循环级，m 是共轭类中元素的数目。如 α 是 GF(2^4)域的 $n=15$ 级元素，则由 $n=15$ 求出 $m=4$，表明该类含 4 个元素；$\beta=\alpha^5$ 是 GF(2^4)域的 3 级元素，则由 $n=3$ 求出 $m=2$，表明该类含 2 个元素。

2. 最小多项式

既约多项式是域理论中的一个重要概念，m 次既约多项式指除了常数外不能被任何幂次低于 m 的因式整除的多项式。然而一个多项式是否既约与多项式的定义域有直接关系，因此还应指明是哪个域上的多项式，即应指明多项式各项系数的定义域。举例说明之。

GF(2^4)域中，15 个非零元素都是 $x^{15}-1=0$ 的根，可写为

$$\alpha^i=\mathrm{e}^{\mathrm{j}2\pi i/15} \qquad (i=0,1,2,\cdots,14) \tag{A-45}$$

于是：

$$x^{15}-1=(x-1)(x-\alpha)(x-\alpha^2)(x-\alpha^3)\cdots(x-\alpha^{14}) \tag{A-46}$$

式中，$(x-\alpha^i)$ 是 GF(2^4)域上定义的多项式，在该域中每个一次因式都是既约因式。

然而在实数域中，复数是不存在的，共轭复数的一对根应合并起来，比如：

$$\begin{aligned}(x-\alpha^3)(x-\alpha^{12})&=(x-\mathrm{e}^{\mathrm{j}6\pi/15})(x-\mathrm{e}^{-\mathrm{j}6\pi/15})\\&=x^2-2x\cos\frac{2\pi}{5}+1\\&=x^2-\frac{\sqrt{5}-1}{4}x+1\end{aligned}$$

$$\begin{aligned}(x-\alpha^6)(x-\alpha^9)&=(x-\mathrm{e}^{\mathrm{j}12\pi/15})(x-\mathrm{e}^{-\mathrm{j}12\pi/15})\\&=x^2-2x\cos\frac{4\pi}{5}+1\\&=x^2+\frac{\sqrt{5}+1}{4}x+1\end{aligned}$$

这样看来，在实数域中最小的既约因式是二次的。

在 GF(2)域中，除了 0 和 1 之外的其他实数都不存在，原属于 GF(2^4)域的同一共轭类元素的一次因式还应进一步合并，如：

$$(x-\alpha^3)(x-\alpha^6)(x-\alpha^9)(x-\alpha^{12})=\left(x^2-\frac{\sqrt{5}-1}{4}x+1\right)\left(x^2+\frac{\sqrt{5}+1}{4}x+1\right)$$
$$=x^4+x^3+x^2+x+1$$

式中，$x^4+x^3+x^2+x+1$ 是定义在 GF(2)域上的 4 次多项式，在 GF(2)域上它已经既约，不能再分解。

域的理论证明了，用同一共轭类各个元素的一次因式连乘，恰好得到的是 GF(2)域的多项式，并且是次数最低的既约多项式。它们都被称为相应域元素的最小多项式。显然，最小多项式的次数就等于它的共轭类中元素的数目。令 $m_i(x)$ 表示元素 α^i 所在共轭类的那个 GF(2)域的多项式，则：

$$m_1(x)=m_2(x)=m_4(x)=m_8(x)=(x-a)(x-a^2)(x-a^4)(x-a^8)$$
$$=x^4+x+1$$
$$m_3(x)=m_6(x)=m_9(x)=m_{12}(x)=(x-a^3)(x-a^6)(x-a^9)(x-a^{12})$$
$$=x^4+x^3+x^2+x+1$$
$$m_5(x)=m_{10}(x)=(x-a^5)(x-a^{10})=x^2+x+1$$
$$m_7(x)=m_{11}(x)=m_{13}(x)=m_{14}(x)=(x-a^7)(x-a^{11})(x-a^{13})(x-a^{14})$$
$$=x^4+x^3+1$$

连同
$$m_0(x)=(x-a^0)=x+1$$
共 5 类。于是在 GF(2)域中，$x^{15}-1$ 因式分解为

$$x^{15}-1=m_0(x)m_1(x)m_3(x)m_5(x)m_7(x) \tag{A-47}$$

作为例子，下面再给出 GF(2^6)域上 63 个非零元素的共轭分类及相应的最小多项式，如表 A.5 所示。

表 A.5 GF(2^6)域的共轭分类及相应的最小多项式

a 的指数	最小多项式	a 的指数	最小多项式
0	$x+1$	13，26，52，41，19，38	$x^6+x^4+x^3+x+1$
1，2，4，8，16，32	x^6+x+1	15，30，60，57，51，39	$x^6+x^5+x^4+x^2+1$
3，6，12，24，48，33	$x^6+x^4+x^2+x+1$	21，42	x^2+x+1
5，10，20，40，17，34	$x^6+x^5+x^2+x+1$	23，46，29，58，53，43	$x^6+x^5+x^4+x+1$
7，14，28，56，49，35	x^6+x^3+1	31，62，61，59，55，47	x^6+x^5+1
9，18，36	x^3+x^2+1	45，27，54	x^3+x+1
11，22，44，25，50，37	$x^6+x^5+x^3+x^2+1$		

3. 本原元素和本原多项式

设 a 是 GF(2^m)域中循环级为 $q-1$ 的元素($q=2^m$)，即 a 自乘 $q-1$ 次又回到群的恒元 1。因为此循环群的阶也是 $q-1$，所以 a 是循环群的单位原根，我们把这样的 a 叫做 GF(2^m)域的一个本原元素。并非域中每个元素都是本原元素，如 GF(2^4)域的循环群为 15 阶，只有 15 级元素 a、a^2、a^4、a^8、a^7、a^{11}、a^{13}、a^{14} 这 8 个元素是本原元素，其他元素都是非本原元素。

如果 GF(2)域上一个 m 次的即约多项式 $m(x)$，它的所有的根都是 GF(2^m)域的本原元素，则 $m(x)$ 被称为本原多项式。换言之，本原元素的最小多项式是本原多项式，非本原

元素的最小多项式是非本原多项式，尽管它们都是既约多项式。

GF(2^4)域中共轭类 a、a^2、a^4、a^8 是本原元素，它们的最小多项式 $m_1(x)=x^4+x+1$ 就被称为本原多项式。共轭类 a^7、a^{11}、a^{13}、a^{14} 也是本原元素，它们的最小多项式 $m_7(x)=x^4+x^3+1$ 也是本原多项式。而其他 3 个最小多项式则是非本原多项式。本原多项式一定是最小多项式，而最小多项式未必是本原多项式。只有本原元素的共轭类为根的最小多项式时才是本原多项式。表 A.6 列出了 $m\leqslant 16$ 次的本原多项式，以供参考。

表 A.6　$m\leqslant 16$ 次的本原多项式

次数	本原多项式	次数	本原多项式
1	$x+1$	9	$x^9+x^8+x^5+x^4+1$
2	x^2+x+1	10	$x^{10}+x^9+x^4+x+1$
3	x^3+x^2+1	11	$x^{11}+x^{10}+x^3+x^2+1$
4	x^4+x^3+1	12	$x^{12}+x^{11}+x^8+x^6+1$
5	$x^5+x^4+x^2+x+1$	13	$x^{13}+x^{12}+x^{10}+x^9+1$
6	x^6+x^5+1	14	$x^{14}+x^{13}+x^8+x^4+1$
7	x^7+x^6+1	15	$x^{15}+x^{14}+1$
8	$x^8+x^7+x^5+x^3+1$	16	$x^{16}+x^{15}+x^{13}+x^4+1$

4. 多项式剩余类的 GF(2^m)域

为了找到 GF(2^m)域各元素之间的运算规律，我们来看 GF(2^m)域的另一个同构表示。

任意给定一个正整数 m，则在 GF(2)域上一定存在一个 m 次的既约多项式：

$$P(x)=c_mx^m+c_{m-1}x^{m-1}+\cdots+c_2x^2+c_1x+c_0 \tag{A-48}$$

在 GF(2)域上存在的多项式，是指该多项式各系数只能取 GF(2)域的元素 0 和 1；既约多项式指除了常数外它不存在幂次低于 m 的因式。

GF(2)域上任意多项式除以 $P(x)$的余式，即幂次低于 m 次的全部多项式，连同能整除时的“零余式”，其数目共 $q=2^m$ 个，它们每一个均可视为以 $P(x)$为模、余数相同的一类多项式的代表，称为多项式剩余类。

定理：m 次本原多项式为模的 2^m 个多项式剩余类的集合是一个 GF(2^m)域。为了证明这一点，只需证明它与一个已知的 GF(2^m)域同构即可。前面已经指出当 $n=2^m-1$ 时，方程 $x^n=1$ 的 n 个根构成 n 阶循环群，再加上一个 0 元素，正好是 GF(2^m)域中的全部成员。现在需要做的工作就是找到这些同余多项式与循环群各元素之间的一一对应关系。

设 a 为 GF(2^m)域的本原元素，则一定也是本原多项式 $P(a)$的根。由 $P(a)=0$ 出发，不难找到 GF(2^m)域的本原元素表达形式与同余多项式表达形式之间的对应关系。以 GF(2^3)域为例，$m=3$，本原多项式为 $P(x)=x^3+x+1$，则由：

$$P(a)=a^3+a+1=0$$

有：

$$a^3=a+1$$

于是：

$$a^4=aa^3=a(a+1)=a^2+a$$

$$a^5=a^2a^3=a^2(a+1)=a^3+a^2=a^2+a+1$$

$$a^6 = a^3 a^3 = (a+1)(a+1) = a^2 + 2a + 1 = a^2 + 1,$$

可见，在模 $P(a)$运算下，$GF(2^3)$域的 8 个元素均表达为幂次不高于 $m=3$ 的多项式形式，见表 A.7。

表 A.7 中不仅列出了本原域元素表达与多项式表达的一一对应关系，还列出了与之对应的 3 位二元序列表达的形式，称为三维矢量表达。至此，关于 $GF(2^3)$域我们有了三种同构的表达形式：本原元素幂表达形式、多项式剩余类表达形式和二元序列表达形式。

表 A.7 $GF(2^3)$ 域 8 个元素的三种表达

本原域元素表达	多项式剩余类表达	二元序列表达
$0 \bmod (a^3+a+1) = 0$	0	(000)
$a^0 \bmod (a^3+a+1) = 1$	1	(001)
$a^1 \bmod (a^3+a+1) = a^1$	x	(010)
$a^2 \bmod (a^3+a+1) = a^2$	x^2	(100)
$a^3 \bmod (a^3+a+1) = a+1$	$x+1$	(011)
$a^4 \bmod (a^3+a+1) = a^2+a$	x^2+x	(110)
$a^5 \bmod (a^3+a+1) = a^2+a+1$	x^2+x+1	(111)
$a^6 \bmod (a^3+a+1) = a^2+1$	x^2+1	(101)
$a^7 \bmod (a^3+a+1) = a^0 = 1$	1	(001)

（$i \geqslant 7$ 以后，又会重复出现与上面相同的元素。）

对于 $m=4$，本原多项式为 $P(x) = x^4 + x + 1$，同样可得到 $GF(2^4)$ 域的 16 个元素的三种表达，见表 A.8。

表 A.8 $GF(2^4)$域的 16 个元素的三种表达

本原域元素	多项式剩余类	二元序列	本原域元素	多项式剩余类	二元序列
0	0	0000	$a^7 = a^3 + a + 1$	$x^3 + x + 1$	1011
$a^0 = a^{15} = 1$	1	0001	$a^8 = a^2 + 1$	$x^2 + 1$	0101
$a^1 = a$	x	0010	$a^9 = a^3 + a$	$x^3 + x$	1010
$a^2 = a^2$	x^2	0100	$a^{10} = a^2 + a + 1$	$x^2 + x + 1$	0111
$a^3 = a^3$	x^3	1000	$a^{11} = a^3 + a^2 + a$	$x^3 + x^2 + x$	1110
$a^4 = a + 1$	$x + 1$	0011	$a^{12} = a^3 + a^2 + a + 1$	$x^3 + x^2 + x + 1$	1111
$a^5 = a^2 + a$	$x^2 + x$	0110	$a^{13} = a^3 + a^2 + 1$	$x^3 + x^2 + 1$	1101
$a^6 = a^3 + a^2$	$x^3 + x^2$	1100	$a^{14} = a^3 + 1$	$x^3 + 1$	1001

由于三种表达是同构的，因此我们可以视其方便，在不同运算中采取不同的表达，如乘法和乘方运算用域元素表达，加减运算用多项式或二进制序列表达。运算结果再用对应关系表达回去。域元素的运算规则只有三条：

（1）循环性，即 $a^{k(q-1)+i} = a^i$，用于乘除运算和降低幂次。

（2）模 $P(x)$运算，即 $P(x) = 0$，用于不同幂次之间的运算。

（3）模 2 加，用于同次幂的加减。

【例】 推导 $GF(2^3)$域和 $GF(2^4)$域的本原多项式。

（1）$GF(2^3)$域。对于本原元素 a（以及 a^2 和 a^4），本原多项式为

$$m_1(x)=(x-a)(x-a^2)(x-a^4)=x^3-(a+a^2+a^4)x^2+(a^3+a^5+a^6)\,x-a^7$$

因为

$$a+a^2+a^4=a+a^2+(a^2+a)=0$$
$$a^3+a^5+a^6=(a+1)+(a^2+a+1)+(a^2+1)=1$$
$$a^7=1$$

所以

$$m_1(x)=x^3+x+1$$

对于本原元素 a^3（以及 a^5 和 a^6），本原多项式为

$$m_3(x)=(x-a^3)(x-a^5)(x-a^6)$$
$$=x^3-(a^3+a^5+a^6)x^2+(a^8+a^9+a^{11})\,x-a^{14}$$

因为

$$a^3+a^5+a^6=(a+1)+(a^2+a+1)+(a^2+1)=1$$
$$a^8+a^9+a^{11}=a+a^2+a^4=0$$
$$a^{14}=1$$

所以

$$m_3(x)=x^3+x^2+1$$

（2）GF(2^4) **域。**

对于本原元素 a（以及 a^2、a^4 和 a^8），本原多项式为

$$m_1(x)=(x-a)(x-a^2)(x-a^4)(x-a^8)$$
$$=x^4-(a+a^2+a^4+a^8)x^3+(a^3+a^5+a^6+a^9+a^{10}+a^{12})x^2$$
$$-(a^7+a^{11}+a^{13}+a^{14})x+a^{15}$$

因为

$$a+a^2+a^4+a^8=0$$
$$a^3+a^5+a^6+a^9+a^{10}+a^{12}=0$$
$$a^7+a^{11}+a^{13}+a^{14}=1$$
$$a^{15}=1$$

所以

$$m_1(x)=x^4+x+1$$

对于本原元素 a^7（以及 a^{11}、a^{13}和 a^{14}），本原多项式为

$$m_7(x)=(x-a^7)(x-a^{11})(x-a^{13})(x-a^{14})$$
$$=x^4-(a^7+a^{11}+a^{13}+a^{14})x^3+(a^{18}+a^{20}+a^{21}+a^{24}+a^{25}+a^{27})x^2$$
$$-(a^{31}+a^{32}+a^{34}+a^{38})x+a^{45}$$

因为

$$a^7+a^{11}+a^{13}+a^{14}=1$$
$$a^{18}+a^{20}+a^{21}+a^{24}+a^{25}+a^{27}=a^3+a^5+a^6+a^9+a^{10}+a^{12}=0$$
$$a^{31}+a^{32}+a^{34}+a^{38}=a+a^2+a^4+a^8=0$$
$$a^{45}=1$$

所以

$$m_7(x)=x^4+x^3+1$$

习 题 A

1. 将 586 表示为七进制数。
2. 用费尔玛算法分解 20 819。
3. 证明模运算法则：若 $a\equiv b \bmod m$，$c\equiv d \bmod m$，则有 $a\pm c\equiv b\pm d \bmod m$。
4. 求 7812 及 6084 的最大公因子。
5. 求解方程 $x^2\equiv 1(\bmod 63)$。
6. 求解方程 $3x\equiv 4(\bmod 121)$。
7. 设奇数 m 是 k 个不同素数之积，证明方程 $x^2\equiv 1(\bmod m)$ 有 2^k 个解。
8. 求方程组 $\begin{cases} x\equiv 5(\bmod 7) \\ x\equiv 2(\bmod 3) \end{cases}$ 的两位数的正整数解。
9. 求 $27^{13}(\bmod 37)$。

附录 B　实践练习的源程序

B.1　Vigenere 密文的生成与破译

B.1.1　Vigenere 的加密程序

自定义函数 transtab[m_, n_]**给出了明文第** n **个字符被加密时所用的置换表(其中，参数** m **应当是以数字列表形式给出的密钥，如** codes **按字母序号可写为**{2, 14, 3, 4, 18})：

```
transtab[m_, n_]:=Table[StringTake[alphabet0, {i}] -> StringTake[abctab,
                    {Mod[i-1+m[[Mod[n-1, Lenth[m]]+1,26]+1]], {i,1,26}];
```

式中：abctab="abcdefghijklmnopqretuvwxyz"

例如，函数返回结果在 m={2, 14, 3, 4, 18}**且** n=1 **时为** a->c, b->d, …, z->b; n=2 **时为** a->o, b->p, …, z->n; n=8 **时为** a->d, b->e, …, z->c。

自定义函数 vigenere[plaintext_, key_]**给出了明文的维吉尼亚加密结果**：

```
vigenere[plaintext_, key_]:=StringJoin[Module[{z}, z={};Do[z=Insert[z,
        StringReplace[Characters[plaintext][[i]], transtab[key, Mod[i-1,
        Length[key]]+1]], i], {i, StringLength[plaintext]}];z]];
```

其中，参数 plaintext **是明文字符串的变量名**；key **是以数字列表形式给出的密钥**。

给参数赋值：

plaintext="**任意选定数百字符的英文并去掉空格**"

Key={**数字列表形式给出的密钥**}

则密文：　　ctxt = vigenere[plaintext, key]

B.1.2　Vigenere 密码的唯密文攻击

1. 密钥长度的判定

自定义函数 coinc[ctxt_, shift_]**用来统计密文中不同距离出现相同字符的次数**：

```
coinc[ctxt_, shift_]:=Module[{z, a, b}, z=0;a=Xharacters[ctxt];b=Drop[a, shift];
                      a=Drop[a, -shift];Do[If[a[[i]]==b[[i]], z=z+1, ]
                      {i, StringLength[ctxt]-shift}]; z];
```

其中，参数 ctxt **是密文变量名**；shift **是欲统计的距离，返回值是次数。分别令** shift=1, 2,

… 计算 coinc[ctxt, shift]，返回值最大的那个 shift 就是最可能的密钥长度 m。

以密钥长度 m 为间距，从密文抽取字符，可得到 m 个子序列。其中第 n 个子序列的自定义函数为

```
choose[ctxtx_, m_, n_]:=StringJoin[Module[{z}, z{ };For[i=1, m*(i-1)+n
                                  < 1+StringLength[ctxt], i++, z=Insert[z,
                                  StringTake[ctxt, {m*(i-1)+n}], i]];z]];
```

2. 密钥的测定

自定义函数 frequency[txt_]用以统计文本 txt 中各字母出现的次数：

```
frequency[txt_]:=Array[Function[i, {Characters[alphabet0][[i]],
                        count[Characters[txt], Characters[alphabet0][[i]]]}, 26];
```

它的返回结果是一个二维序列：$\{a, n_1\}, \{b, n_2\}, \cdots, \{z, n_{26}\}$。

统计从密文抽取的第 n 个子序列中各字母出现频度的自定义函数为

```
Vigvec[ctxt_, m_, n_]:=Module[(z, w), z=choose[ctxt, m, n];
      w=Table[frequency[z][[i]][[2]]/StringLenth[z]*1., {I, 1, 26}];
```

它的返回值是一个 26 列的向量，给出字母 a～z 出现的相对频度。例如，当计算第一个子序列中各字母的频度矢量时，可引入变量：

```
frevec1= Vigvec[ctxt, m, 1]
```

计算一个矢量 a 与另一个矢量 b 的位移矢量的点乘积的自定义函数为

```
s[a_, b_, n_]:=Sum[a[[i]]*b[[Mod[i-n-1, 26]+1]], {I, 26}];
```

其中，参数 n 表示矢量 b 循环位移的位数。

由此可以写出自定义函数 corr[subctxt_]，用来计算从密文抽取的第 n 个子序列中各字母的频度矢量与标准英语字母频度的位移矢量的点乘积，并按位移数值把各个点积排成一个列表，叫做相关函数矢量。

```
corr[frevecn_]:=Table[s[frevecn, alfreq, i], {i, 0, 25}];
```

式中：

```
alfreq={0.082, 0.015, 0.028, 0.043, 0.127, 0.022, 0.020, 0.061, 0.070,
        0.002, 0.008, 0.040, 0.024, 0.067, 0.075, 0.019, 0.001, 0.060,
        0.063, 0.091, 0.028, 0.010, 0.023, 0.001, 0.020, 0.001}
```

是标准英语字母出现频度的列表。

相关函数矢量的最大分量，是因为有最恰当的位移值所致，这个位移值就是该子序列加密时使用的位移值 k_n；分别计算出 n=1, 2, …, m 各个子序列的位移值 k_n，则 vigenere 密钥为

$$k=\{k_1, k_2, \cdots, k_m\}$$

3. 密文的解译

密文解译时仍使用 vigenere 加密程序，只不过做反向移位即可：

```
mtxt= vigenere[ctxt , - k]
```

B.2 *m* 序列密码系统的已知部分明文攻击

已知 40 位密文序列，但只知道 17 位明文：

```
ctxt={0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
      1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0}
mtxt={1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0}
```

要求破译全部密文并且破解生成密钥序列的线性反馈移位寄存器。

(1) 由密文与部分明文按位模 2 加，求出部分密钥 key：

```
Key=Table[Mod[mtxt[[i]]+ctxt[[i]], 2], {i, 1, Length[mtxt]}]
```

(2) 由部分 m 序列来确定线性反馈移位寄存器的级数：

```
lfsrlenth[key_, n_]:=Do[Print[{k, Mod[Array[Function[{i, j}, key[[i+j-1]],
                    {k, k}]], 2]}], {k, Min[n, 1+Floor[Dimension[key][[1]]/2]]}];
```

它的返回结果是一个二维列表：{1, x}，{2, x}，{3, x}，…，{n, x}，分别给出由密钥序列写出的不同维数的错位方阵的模 2 行列式值；n 是预先设定维数的界限，x 可以取 0 或 1。观察这些 x 值，前面维数较低的 x 值有 0 有 1，后面的 x 就只有 0 了。找到最后那个等于 1 的 x，它的维数 t 就是能产生这个 m 序列的线性反馈移位寄存器的级数。

(3) 寻找线性反馈移位寄存器的特征多项式。由已经确定级数的错位方阵的逆矩阵乘以密钥序列的下一个错位节，就可以得到线性反馈移位寄存器的特征多项式。计算它的自定义函数为

```
lfsrsolve[key_, n_]:=Module[{z}, z=Array[Function[[{i, j}, key[[i+j-1]]], {n, n}];
                     Mod[Det[z]*Inverse[z].Array[Function[i, key[[i+n]]], n], 2]];
```

它的返回值为 LFSR 特征多项式的系数 $\{c_0, c_1, c_2, \cdots, c_{n-1}\}$，**可以将其赋值给变量** c：

```
c=lfsrsolve[key , t]
```

(4) 生成任意长度的 m 序列的自定义函数，为

```
lfsr[c_, k_, n_]:=Module[{z}, z=k;Do[z=Array[z, Mod[Array[Function[i, z[[j-
                  [Length[k]-1+i]]], Length[k]].c, 2]], {j, Length[k]+1, n}];z];
```

式中，c 是特征多项式的系数列表；k 是寄存器的初始状态；n 是设定的 m 序列输出长度。

(5) 解密全部密文。截取与密文相同长度的 m 序列按位模 2 加，而 m 序列可调用 lfsr[c_, k_, n_]：

```
Decryp[ctxt_, c_, k_]:=Table[Mod[ctxt[[i]]+lfsr[c, k, Length[ctxt]][[i]], 2],
                       {i, 1, Length[ctxt]}];
```

B.3　DES 分组加密与解密的源程序

本程序根据 Bruce Schneier **所著的《应用密码学》[5]的附录 A.1 改编得到。加密与解密都是同一个程序，回答“e”与“d”加以区别。加密时键盘输入 8 个字符的密钥，输入明文文件名，密文自动存入“outf.txt”的文件中。解密时输入同样的密钥，回答准备存储译文的文件名，密文解译后自动存入其中。**

DES **的 C 语言源程序**(des.c)：

```
#include "des.h"
void deskey(unsigned char * key, short edf, unsigned long * keystr)
```

```
{
    register int i, j, l, m, n;
    unsigned char pclm[56], pcr[56];
    unsigned long kn[32]={0L};
    unsigned long *raw0, *cook, *raw1;
    cook=keystr;
    raw1=kn;
    for(j=0;j<56;j++)
    {
        l=pc1[j];
        m=l&07;
        pclm[j]=(key[l>>3]&bytebit[m])?1:0;
    }
    for(i=0;i<16;i++)
    {
        if(edf==DE1) m=(15-i)<<1;
        else m=i<<1;
        n=m+1;
        kn[m]=kn[n]=0L;
        for(j=0;j<28;j++)
        {
            l=j+totrot[i];
            if(l<28) pcr[j]=pclm[l];
            else pcr[j]=pclm[l-28];
        }
    for(j=28;j<56;j++){
            l=j+totrot[i];
            if(l<56) pcr[j]=pclm[l];
            else pcr[j]=pclm[l-28];
        }
    for(j=0;j<24;j++) {
        if( pcr[pc2[j]]) kn[m]|=bigbyte[j];
        if( pcr[pc2[j+24]]) kn[n]|=bigbyte[j];
        }
    }
        for(i=0;i<16;i++, raw1++) {
        raw0=raw1++;
        *cook =(*raw0       & 0x00fc0000L)<<6;
        *cook |=(*raw0      & 0x00000fc0L)<<10;
        *cook |=(*raw1      & 0x00fc0000L)>>10;
        *cook++|=(*raw1 & 0x00000fc0L)>>6;
        *cook =(*raw0       & 0x0003f000L)<<12;
        *cook |=(*raw0      & 0x0000003fL)<<16;
```

```
            *cook |=(*raw1      & 0x0003f000L)>>4;
            *cook++|=(*raw1 & 0x0000003fL);
        }
        return;
}

static void scrunch(register unsigned char *outof, register unsigned long *into){
        *into =(*outof++        & 0xffL)<<24;
        *into |=(*outof++       & 0xffL)<<16;
        *into |=(*outof++       & 0xffL)<<8;
        *into++|=(*outof++ & 0xffL);
        *into =(*outof++        & 0xffL)<<24;
        *into |=(*outof++       & 0xffL)<<16;
        *into |=(*outof++       & 0xffL)<<8;
        *into |=(*outof         & 0xffL);

}
static void unscrun(register unsigned long *outof, register unsigned char *into)
{
        *into++=(unsigned char)((*outof>>24) & 0xffL);
        *into++=(unsigned char)((*outof>>16) & 0xffL);
        *into++=(unsigned char)((*outof>> 8) & 0xffL);
        *into++=(unsigned char)(*outof++        & 0xffL);
        *into++=(unsigned char)((*outof>>24) & 0xffL);
        *into++=(unsigned char)((*outof>>16) & 0xffL);
        *into++=(unsigned char)((*outof>> 8) & 0xffL);
        *into++=(unsigned char)(*outof          & 0xffL);
        return;
}
static void desfunc(register unsigned long *block, register unsigned long *keys,
                    register unsigned long *mcipher)
{
        register unsigned long fval, work, right, leftt;
        register int round;
        leftt=block[0];
        right=block[1];
        work=((leftt>>4)^right) & 0x0f0f0f0fL;
        right^=work;
        leftt^=(work<<4);
        work=((leftt>>16)^right) & 0x0000ffffL;
        right^=work;
        leftt^=(work<<16);
        work=((right>>2)^leftt) & 0x33333333L;
```

```
leftt^=work;
right^=(work<<2);
work=((right>>8)^leftt) & 0x00ff00ffL;
leftt^=work;
right^=(work<<8);
right=((right<<1)|((right>>31)&1L)) & 0xffffffffL;
work=(leftt^right) & 0xaaaaaaaaL;
leftt^=work;
right^=work;
leftt=((leftt<<1)|((leftt>>31)&1L)) & 0xffffffffL;
for(round=0;round<8;round++){
    work =(right<<28)|(right>>4);
    work^= *keys++;
    fval =sp7[work & 0x3fL];
    fval|=sp5[(work>>8)&0x3fL];
    fval|=sp3[(work>>16)&0x3fL];
    fval|=sp1[(work>>24)&0x3fL];
    work =right^*keys++;
    fval|=sp8[work & 0x3fL];
    fval|=sp6[(work>>8)&0x3fL];
    fval|=sp4[(work>>16)&0x3fL];
    fval|=sp2[(work>>24)&0x3fL];
    leftt^=fval;
    work =(leftt<<28)|(leftt>>4);
    work^= *keys++;
    fval =sp7[work & 0x3fL];
    fval|=sp5[(work>>8)&0x3fL];
    fval|=sp3[(work>>16)&0x3fL];
    fval|=sp1[(work>>24)&0x3fL];
    work =leftt^*keys++;
    fval|=sp8[work & 0x3fL];
    fval|=sp6[(work>>8)&0x3fL];
    fval|=sp4[(work>>16)&0x3fL];
    fval|=sp2[(work>>24)&0x3fL];
    right^=fval;
}
right=(right<<31)|(right>>1);
work=(leftt^right) & 0xaaaaaaaaL;
leftt^=work;
right^=work;
leftt=(leftt<<31)|(leftt>>1);
work=((leftt>>8)^right) & 0x00ff00ffL;
right^=work;
```

```
    leftt^=(work<<8);
    work=((leftt>>2)^right) &0x33333333L;
    right^=work;
    leftt^=(work<<2);
    work=((right>>16)^leftt) &0x0000ffffL;
    leftt^=work;
    right^=(work<<16);
    work=((right>>4)^leftt) &0x0f0f0f0fL;
    leftt^=work;
    right^=(work<<4);
    *mcipher++=right;
    *mcipher=leftt;
    return;
}

void des_enc(unsigned char *fkey, unsigned char *data, unsigned char *cipher)
{
    unsigned long work[2];
    unsigned long mkey[32]={0L}, mcipher[2]={0L};
    deskey(fkey, EN0, mkey);
    scrunch(data, work);
    desfunc(work, mkey, mcipher);
    unscrun(mcipher, cipher);
}

void des_dec(unsigned char *fkey, unsigned char *cipher, unsigned char *plaint){
        unsigned long work[2];
        unsigned long mkey[32]={0L}, mplaint[2]={0L};
        deskey(fkey, DE1, mkey);
        scrunch(cipher, work);
        desfunc(work, mkey, mplaint);
        unscrun(mplaint, plaint);
    }

void main(void)
{
    int i;
    FILE *fp;
    FILE *outfp;
    char ch, f_name[20];
    unsigned char data[8]={0};
    unsigned char key[8];
    unsigned char cipher[8]={0L}, plaint[8]={0L};
```

```
    printf("Input your key:");
    gets(key);
star:
    printf("Whether encryp(e) or decryp(d) ?");
    ch=getch();
    if(ch=='e') {
      printf("\nInput plaintTEXT filename:");
      gets(f_name);
        if((fp=fopen(f_name, "r"))==NULL) {
        printf("can't open file\n");
        return;
        }
      if((outfp=fopen("outf.txt", "wb"))==NULL) {
        printf("can't open file\n");
        return;
        }
    while(!feof(fp)) {
      for(i=0;i<8;i++) {
        if(feof(fp)) data[i]=0x20;
        data[i]=getc(fp);
        }
      des_enc(key, data, cipher);
      fwrite(&cipher, 1, 8, outfp);
      }
    }
    else if(ch=='d') {
      if((outfp=fopen("outf.txt", "rb"))==NULL) {
        printf("can't open file\n");
        return;
        }
      printf("\nInput TEXT filename:");
      gets(f_name);
      if((fp=fopen(f_name, "w"))==NULL) {
        printf("can't open file\n");
        return;
        }
      while(!feof(outfp)) {
        fread(&cipher, 1, 8, outfp);
        des_dec(key, cipher, plaint);
        for(i=0;i<8;i++) {
          if(feof(outfp)) break;
          putc(plaint[i], fp);
```

```
            }
          }
        }
    else goto star;
    fclose(fp);
    fclose(outfp);
    getch();
}
```

(2) **头文件**(des.h):

```
#define EN0 0
#define DE1 1
#include<stdio.h>

extern void deskey(unsigned char *, short, unsigned long *);
static void scrunch(unsigned char *, unsigned long *);
static void unscrun(unsigned long *, unsigned char *);
static void desfunc(unsigned long *, unsigned long *, unsigned long *);
static void cookey(unsigned long *, unsigned long *);

static unsigned short bytebit[8] ={0200, 0100, 040, 020, 010, 04, 02, 01};
static unsigned long bigbyte[24] =
{
    0x800000L, 0x400000L, 0x200000L, 0x100000L,
    0x80000L, 0x40000L, 0x20000L, 0x10000L,
    0x8000L, 0x4000L, 0x2000L, 0x1000L,
    0x800L, 0x400L, 0x200L, 0x100L,
    0x80L, 0x40L, 0x20L, 0x10L,
    0x8L, 0x4L, 0x2L, 0x1L};
    static unsigned char pc1[56] =
    {
    56, 48, 40, 32, 24, 16, 8, 0, 57, 49, 41, 33, 25, 17,
    9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35,
    62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21,
    13, 5, 60, 52, 44, 36, 28, 20, 12, 4, 27, 19, 11, 3};
static unsigned char totrot[16] ={1, 2, 4, 6, 8, 10, 12, 14, 15, 17, 19, 21, 23, 25, 27, 28};
static unsigned char pc2[48] =
{
    13, 16, 10, 23,  0,  4,  2, 27, 14, 5, 20,  9,
    22, 18, 11,  3, 25,  7, 15,  6, 26, 19, 12,  1,
    40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
    43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31, };

static unsigned long sp1[64] =
```

```
{
    0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
    0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
    0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
    0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
    0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
    0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
    0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
    0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
    0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
    0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
    0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
    0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
    0x01000004L, 0x00000404L, 0x00010404L, 0x01010404L,
    0x00000404L, 0x01000404L, 0x01000400L, 0x00000000L,
    0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L, };
static unsigned long sp2[64] =
{
    0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
    0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
    0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
    0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
    0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
    0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
    0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
    0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
    0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
    0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
    0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
    0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
    0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
    0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
    0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
    0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L, };
static unsigned long sp3[64] =
{
    0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
    0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
    0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
    0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
    0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
    0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
```

```
    0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
    0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
    0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
    0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
    0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
    0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
    0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
    0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
    0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
    0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L};
static unsigned long sp4[64] =
{
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
    0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
    0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
    0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
    0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
    0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
    0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
    0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
    0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
    0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
    0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
    0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
    0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L};

static unsigned long sp5[64] =
{
    0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
    0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
    0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
    0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
    0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
    0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
    0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
    0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
    0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
    0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
    0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
    0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
```

```
    0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
    0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
    0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
    0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L};

static unsigned long sp6[64] =
{
    0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
    0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
    0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
    0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
    0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
    0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
    0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
    0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
    0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
    0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
    0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
    0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
    0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
    0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
    0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
    0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L};

static unsigned long sp7[64] =
{
    0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
    0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
    0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
    0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
    0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
    0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
    0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
    0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
    0x04000000L, 0x00200800L, 0x00200000L, 0x04200802L,
    0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
    0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
    0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
    0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
    0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
    0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
    0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L};
```

```
static unsigned long sp8[64] =
{
    0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
    0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
    0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
    0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
    0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
    0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
    0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
    0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
    0x00041040l, 0x00040000L, 0x10041000L, 0x00001000L,
    0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
    0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
    0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
    0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
    0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
    0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
    0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L};
```

B.4　RSA 公开密钥体系的构建与加密解密

1. 大素数的产生与检测

Mathematica 4.0 **软件有内置的数论函数包，可以用下面的命令装载：**

```
<<NumberTheory 'NumberTheoryFunctions'
```

产生一个十进制 100 **位数的随机素数的函数是：**

```
p=NextPrime[Random[Integer, {10^99, 10^100}]]
```

检测一个数是不是素数的命令是：

```
PrimeQ[p]
```

2. RSA 密码体制的构建

产生一个 129 **位的大素数** p **和一个** 131 **位的大素数** q，**计算** n=p*q **和** phi=(p−1)*(q−1)。**再产生一个** 50 **位的素数** e **为公钥，计算** d=PowerMod[e, −1, n]**为私钥。**

3. 将已知的明文变换成数字形式

明文数字化方式很多，为了简单，这里采用英语字母表的序号，空格序号为 00 **并忽略标点符号。自定义的转换函数为**

```
num1[plaintext_] := ToExpression[StringReplace[plaintext, corresp]];
```

式中：

```
corresp=Table[StringTake[abctab1, {i}]->StringTake[numbers1,
        {2*i-1, 2*i}], {i, 1, 27}];
abctab1="abcdefghijklmnopqrstuvwxyz";
```

numbers1="000102030405060708091011121314151617181920212223242526"

4. RSA 体制的加密与解密

把明文分割成 20 **个字符左右的段落(长度可以不完全相等),分别赋值给变量** mtext1、mtext2 … **每次加密一段**:

ctex1=PowerMod[num1[mtext1], e, n]

解密仍然分段进行:

decryptext=alph1[PowerMod[ctext1, e, n]]

式中, alph1[numbers]**是将数字变回字母的转换函数,自定义形式为**

```
alph1[n_]:=Module[{c, x, z}, x=n;c=Characters[alphabet1];z={};
        While[x>0, z=Prepend[z, c[[Mod[x, 100]+1]]]];
        x=Floor[x/100]];StringJoin[z]];
```

5. 离散对数密码体制

产生一个 40 **位的大素数** p,**并求出它的本原根**:

g=PrimitiveRoot[p];

产生随机数 a =Random[Integer, {10^15, 10^16}]**为私钥,计算** b =PowerMod[g, a, p],**则**{g, p, b }**都是公钥**。

加密时首先产生一个随机数:

k= Random[Integer, {10^18, 10^19}]

设明文段落(小于 20 **个字母)储存在变量** mtxt **中,数字化为** m=num1[mtxt],**则密文由两部分组成**:

```
y1=PowerMod[g, k, p];
y2=Mod[m * PowerMod[b, k, p], p];
```

解密时只需计算:

```
Mod[y2 * PowerMod[y1, -a, p], p];
```

B.5 MD5 信息摘要进行数字签名的安全通信

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
(1) 演示给文本填加数字签名的程序。
    从一个 txt 文本文件中读取全部字符,计算其 Hash 值,得到 512 bit (32 位十六进制数)
的信息摘要。然后输入私钥将其加密得到签名,将此签名写在文本末尾,以原文件名存盘。
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#include "stdio.h"
#include "math.h"
static unsigned long aa, bb, cc, dd;
static char e[32767];
FILE *fp;
```

```
/*读取文件数据，按512 bit长的二进数分段，为计算摘要做好准备*/
int addup()
   {
     char ch, b[4];
     int i, j;
     unsigned long l, m, n;
     int count;
     i=0;
     ch=getc(fp);
     while(ch!=EOF) {
       e[i]=ch;
       ch=getc(fp);
       i++;
       }
fclose(fp);
   printf("In this text the number of all characters is:");
   printf("%d\n", i);
   count=i*8;
   m=count%512;
j=1;
e[i]=128;
if(m<480) {
   while(j<((480-m)/8))
     {i++;e[i]=00;j++;}
     }
else if(m>480){
   while(j<((992-m)/8))
     {i++;e[i]=00;j++;}
     }
   else if(m==480){
     while(j<(512/8))
     {i++;e[i]=00;j++;}
      }
   m=(i+1)*8;
   n=0xff000000;
   l=m&n;
   b[3]=(l>>24);
   n=0x00ff0000;
   l=m&n;
   b[2]=(l>>16);
   n=0x0000ff00;
   l=m&n;
   b[1]=(l>>8);
```

```
  n=0x000000ff;
  b[0]=m&n;
  j=0;
  while(j<4)
    {e[++i]=b[j];
    j++; }
  return(i);
   }

/* MD5 所要求的非线性运算子程序 */
unsigned long ff(unsigned long x, unsigned long y, unsigned long z)
   { unsigned long m, n;
     m=~x;
     m=m&z;
     n=x&y;
     n=m|n;
     return(n);
  }
unsigned long gg(unsigned long x, unsigned long y, unsigned long z)
  { unsigned long m, n;
    m=~z;
    m=m&y;
    n=x&z;
    n=m|n;
    return(n);
  }
unsigned long hh(unsigned long x, unsigned long y, unsigned long z)
   { unsigned long n;
     n=x^y;
     n=n^z;
     return(n);
  }
unsigned long ii(unsigned long x, unsigned long y, unsigned long z)
   { unsigned long n;
     n=~z;
     n=n|x;
     n=n^y;
     return(n);
   }
/* MD5 所要求的置换与循环子程序 */
unsigned long round1(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
unsigned long l, int m, unsigned long n)
   { unsigned long o, p, q;
```

```
    unsigned long ff(unsigned long x, unsigned long y, unsigned long z);
    p=ff(i, j, k);
    p=p+h+l+n;
    o=p>>(32-m);
    q=p<<m;
    p=o|q;
    p=p+i;
    return(p);
  }
unsigned long round2(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
unsigned long l, int m, unsigned long n)
  { unsigned long o, p, q;
    unsigned long gg(unsigned long x, unsigned long y, unsigned long z);
    p=gg(i, j, k);
    p=p+h+l+n;
    o=p>>(32-m);
    q=p<<m;
    p=o|q;
    p=p+i;
    return(p);
  }
unsigned long round3(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
unsigned long l, int m, unsigned long n)
  { unsigned long o, p, q;
    unsigned long hh(unsigned long x, unsigned long y, unsigned long z);
    p=hh(i, j, k);
    p=p+h+l+n;
    o=p>>(32-m);
    q=p<<m;
    p=o|q;
    p=p+i;
    return(p);
  }
unsigned long round4(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
unsigned long l, int m, unsigned long n)
  { unsigned long o, p, q;
    unsigned long ii(unsigned long x, unsigned long y, unsigned long z);
    p=ii(i, j, k);
    p=p+h+l+n;
    o=p>>(32-m);
    q=p<<m;
    p=o|q;
    p=p+i;
```

```
    return(p);
  }

/* 计算信息摘要的 MD5 程序 */
void md5(void)
  { int i, j, k, m;
    unsigned long x[16];
    unsigned long t[65];
    unsigned long a, b, c, d;
    union long_char
    { unsigned long l;
      char ch[4];
      } y;
    int addup(void);
    unsigned long round1(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
                         unsigned long l, int m, unsigned long n);
    unsigned long round2(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
                         unsigned long l, int m, unsigned long n);
    unsigned long round3(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
                         unsigned long l, int m, unsigned long n);
    unsigned long round4(unsigned long h, unsigned long i, unsigned long j, unsigned long k,
                         unsigned long l, int m, unsigned long n);

    m=addup();
    aa=0x01234567;
    bb=0x89abcdef;
    cc=0xfedcba98;
    dd=0x76543210;
    k=0;
    while(k<((m+1)/512)) {
      for(i=0;i<16;i++)
        { for(j=0;j<4;j++) y.ch[j]=e[k*64+i*4+j];
          x[i]=y.l;
        }
      a=aa;
      b=bb;
      c=cc;
      d=dd;
    t[1]=0xd76aa478;t[2]=0xe8c7b756;t[3]=0x242070db;t[4]=0xc1bdceee;
    t[5]=0xf57c0faf;t[6]=0x4787c62a;t[7]=0xa8304613;t[8]=0xfd469501;
    t[9]=0x698098d8;t[10]=0x8b44f7af;t[11]=0xffff5bb1;t[12]=0x895cd7be;
    t[13]=0x6b901122;t[14]=0xfd987193;t[15]=0xa679438e;t[16]=0x49b40821;
    t[17]=0xf61e2562;t[18]=0xc040b340;t[19]=0x265e5a51;t[20]=0xe9b6c7aa;
```

```
t[21]=0xd62f105d;t[22]=0x02441453;t[23]=0xd8a1e681;t[24]=0xe7d3fbc8;
t[25]=0x21e1cde6;t[26]=0xc33707d6;t[27]=0xf4d50d87;t[28]=0x455a14ed;
t[29]=0xa9e3e905;t[30]=0xfcefa3f8;t[31]=0x676f02d9;t[32]=0x8d2a4c8a;
t[33]=0xfffa3942;t[34]=0x8771f681;t[35]=0x699d6122;t[36]=0xfde5380c;
t[37]=0xa4beea44;t[38]=0x4bdecfa9;t[39]=0xf6bb4b60;t[40]=0xbebfbc70;
t[41]=0x289b7ec6;t[42]=0xeaa127fa;t[43]=0xd4ef3085;t[44]=0x04881d05;
t[45]=0xd9d4d039;t[46]=0xe6db99e5;t[47]=0x1fa27cf8;t[48]=0xc4ac5665;
t[49]=0xf4292244;t[50]=0x432aff97;t[51]=0xab9423a7;t[52]=0xfc93a039;
t[53]=0x655b59c3;t[54]=0x8f0ccc92;t[55]=0xffeff47d;t[56]=0x85845dd1;
t[57]=0x6fa87e4f;t[58]=0xff2ce6ed;t[59]=0xa3014314;t[60]=0x4e0811a1;
t[61]=0xf7537e82;t[62]=0xbd3af235;t[63]=0x2ad7d2bb;t[64]=0xeb86d391;
a=round1(a, b, c, d, x[0], 7, t[1]);
d=round1(d, a, b, c, x[1], 12, t[2]);
c=round1(c, d, a, b, x[2], 17, t[3]);
b=round1(b, c, d, a, x[3], 22, t[4]);
a=round1(a, b, c, d, x[4], 7, t[5]);
d=round1(d, a, b, c, x[5], 12, t[6]);
c=round1(c, d, a, b, x[6], 17, t[7]);
b=round1(b, c, d, a, x[7], 22, t[8]);
a=round1(a, b, c, d, x[8], 7, t[9]);
d=round1(d, a, b, c, x[9], 12, t[10]);
c=round1(c, d, a, b, x[10], 17, t[11]);
b=round1(b, c, d, a, x[11], 22, t[12]);
a=round1(a, b, c, d, x[12], 7, t[13]);
d=round1(d, a, b, c, x[13], 12, t[14]);
c=round1(c, d, a, b, x[14], 17, t[15]);
b=round1(b, c, d, a, x[15], 22, t[16]);
a=round2(a, b, c, d, x[1], 5, t[17]);
d=round2(d, a, b, c, x[6], 9, t[18]);
c=round2(c, d, a, b, x[11], 14, t[19]);
b=round2(b, c, d, a, x[0], 20, t[20]);
a=round2(a, b, c, d, x[5], 5, t[21]);
d=round2(d, a, b, c, x[10], 9, t[22]);
c=round2(c, d, a, b, x[15], 14, t[23]);
b=round2(b, c, d, a, x[4], 20, t[24]);
a=round2(a, b, c, d, x[9], 5, t[25]);
d=round2(d, a, b, c, x[14], 9, t[26]);
c=round2(c, d, a, b, x[3], 14, t[27]);
b=round2(b, c, d, a, x[8], 20, t[28]);
a=round2(a, b, c, d, x[13], 5, t[29]);
d=round2(d, a, b, c, x[2], 9, t[30]);
c=round2(c, d, a, b, x[7], 14, t[31]);
b=round2(b, c, d, a, x[12], 20, t[32]);
```

```
    a=round3(a, b, c, d, x[5], 4, t[33]);
    d=round3(d, a, b, c, x[8], 11, t[34]);
    c=round3(c, d, a, b, x[11], 16, t[35]);
    b=round3(b, c, d, a, x[14], 23, t[36]);
    a=round3(a, b, c, d, x[1], 4, t[37]);
    d=round3(d, a, b, c, x[4], 11, t[38]);
    c=round3(c, d, a, b, x[7], 16, t[39]);
    b=round3(b, c, d, a, x[10], 23, t[40]);
    a=round3(a, b, c, d, x[13], 4, t[41]);
    d=round3(d, a, b, c, x[0], 11, t[42]);
    c=round3(c, d, a, b, x[3], 16, t[43]);
    b=round3(b, c, d, a, x[6], 23, t[44]);
    a=round3(a, b, c, d, x[9], 4, t[45]);
    d=round3(d, a, b, c, x[12], 11, t[46]);
    c=round3(c, d, a, b, x[15], 16, t[47]);
    b=round3(b, c, d, a, x[2], 23, t[48]);
    a=round4(a, b, c, d, x[0], 6, t[49]);
    d=round4(d, a, b, c, x[7], 10, t[50]);
    c=round4(c, d, a, b, x[14], 15, t[51]);
    b=round4(b, c, d, a, x[5], 21, t[52]);
    a=round4(a, b, c, d, x[12], 6, t[53]);
    d=round4(d, a, b, c, x[3], 10, t[54]);
    c=round4(c, d, a, b, x[10], 15, t[55]);
    b=round4(b, c, d, a, x[1], 21, t[56]);
    a=round4(a, b, c, d, x[8], 6, t[57]);
    d=round4(d, a, b, c, x[15], 10, t[58]);
    c=round4(c, d, a, b, x[6], 15, t[59]);
    b=round4(b, c, d, a, x[13], 21, t[60]);
    a=round4(a, b, c, d, x[4], 6, t[61]);
    d=round4(d, a, b, c, x[11], 10, t[62]);
    c=round4(c, d, a, b, x[2], 15, t[63]);
    b=round4(b, c, d, a, x[9], 21, t[64]);
    aa=a+aa;
    bb=b+bb;
    cc=c+cc;
    dd=d+dd;
    k++;
    }
    printf("\The MD5 abstract of this text is:\n");
    printf("%lx, %lx, %lx, %lx\n", aa, bb, cc, dd);
}
```

/* **加密运算：计算模** nn **运算下** aa **的** bb **次方** */

```
unsigned long encode(unsigned long aa, unsigned long bb, unsigned long nn)
  { int i, k;
    unsigned long cc, dd;
    int bbb[32];
    dd=0x80000000;
    i=31;
    while(i>=0)
      { cc=dd&bb;
        cc=cc>>i;
        if(cc==1)
          {k=i;break;}
        else
          {i--;dd=(dd>>1);}
          }
  dd=0x80000000;
  i=31;
  while(i>=0)
    {cc=dd&bb;
    cc=cc>>i;
      if(cc==1)
        {bbb[i]=1;i--;dd=(dd>>1);}
      else
        {bbb[i]=0;i--;dd=(dd>>1);}
    }
  dd=1;
  while(k>=0)
    {
      dd=(dd*dd)%nn;
      if(bbb[k]==1)
        {dd=(dd*aa)%nn;}
        k--;
    }

    return(dd);
  }

/*--------------------主程序--------------------*/
main(){
  union long_char
    { unsigned long ceo;
      unsigned char cfo[4];
    }kof;
  union long_int
```

```
  { unsigned int cef;
    unsigned char cff[2];
    }kff;
unsigned int cai, pai;
unsigned int hai[16];
int j=0;
char f_name[20];
void md5(void);
unsigned long encode(unsigned long aa, unsigned long bb, unsigned long nn);
printf("Input filename:\n");
gets(f_name);
  if((fp=fopen(f_name, "r"))==NULL)
  { printf("can't open file\n");
    return 1;
  }
md5();
printf("Input your private key: 'd, n':");
scanf("%ld, %ld", &cai, &pai);
kof.ceo=aa;
hai[0]=encode(kof.cfo[0], cai, pai);
hai[1]=encode(kof.cfo[1], cai, pai);
hai[2]=encode(kof.cfo[2], cai, pai);
hai[3]=encode(kof.cfo[3], cai, pai);

kof.ceo=bb;
hai[4]=encode(kof.cfo[0], cai, pai);
hai[5]=encode(kof.cfo[1], cai, pai);
hai[6]=encode(kof.cfo[2], cai, pai);
hai[7]=encode(kof.cfo[3], cai, pai);

kof.ceo=cc;
hai[8]=encode(kof.cfo[0], cai, pai);
hai[9]=encode(kof.cfo[1], cai, pai);
hai[10]=encode(kof.cfo[2], cai, pai);
hai[11]=encode(kof.cfo[3], cai, pai);

kof.ceo=dd;
hai[12]=encode(kof.cfo[0], cai, pai);
hai[13]=encode(kof.cfo[1], cai, pai);
hai[14]=encode(kof.cfo[2], cai, pai);
hai[15]=encode(kof.cfo[3], cai, pai);
if((fp=fopen(f_name, "a"))==NULL)
  { printf("can't open file\n");
```

```
        return 1;
      }
    printf("The result after encrypting is:\n");
    j=0;
    while(j<16)
      { printf("%x, ", hai[j]);
        kff.cef=hai[j];
        putc(kff.cff[0], fp);
        putc(kff.cff[1], fp);
        j++;
      }
    printf("\n");
    fclose(fp);
}
/*************************************************
```

(2) **演示验证数字签名的程序。**

从一个带有数字签名的 txt 文件中读取文本，计算其 Hash 值，得到 512 bit(32 位十六进制数)的信息摘要。然后输入公钥，将文件末尾的数字签名解密，将解密的结果与计算得到的信息摘要比较，二者一致即可证明有效。

```
*************************************************/
#include "stdio.h"
#include "math.h"
static unsigned long aa, bb, cc, dd;
static unsigned int hai[16];
static char e[32767];

/*读取文件数据，按 512 bit 长的二进数分段，并把末尾的计算摘要分离出来*/
int addup()
   { union long_char
       { unsigned int cef;
         unsigned char cff[2];
       } kff;
     char ch, b[4], f_name[20];
     int i, j;
     unsigned long l, m, n;
     int count;
     FILE *fp;
     printf("Input filename:\n");
     gets(f_name);
     if((fp=fopen(f_name, "r"))==NULL)
       { printf("can't open file\n");
         return 1;
       }
```

```
i=0;
ch=getc(fp);
while(ch !=EOF) {
   e[i]=ch;
   ch=getc(fp);
   i++;
   }
   fclose(fp);
    j=15;
while(j>=0){
        i=i-1;
        kff. cff[1]=e[i];
        e[i]=00;
        i=i-1;
        kff. cff[0]=e[i];
        e[i]=00;
        hai[j]=kff. cef;
        j--;
     }
j=0;
printf("The encrypted signature of this text is:\n");
while(j<=15)
   {printf("%x, ", hai[j]);j++;}
   printf("\n");
      printf("In this text the number of all characters is:");
      printf("%d\n", i);
count=i*8;
m=count%512;
j=1;
e[i]=128;
if(m<480) {
   while(j<((480-m)/8))
   {i++;e[i]=00;j++;}
   }
else if(m>480){
   while(j<((992-m)/8))
     {i++;e[i]=00;j++;}
   }
else if(m==480){
   while(j<(512/8))
     {i++;e[i]=00;j++;}
   }
```

```
    m=(i+1)*8;
    n=0xff000000;
    l=m&n;
    b[3]=(l>>24);
    n=0x00ff0000;
    l=m&n;
    b[2]=(l>>16);
    n=0x0000ff00;
    l=m&n;
    b[1]=(l>>8);
    n=0x000000ff;
    b[0]=m&n;
    j=0;
    while(j<4)
      {e[++i]=b[j];
      j++; }
    return(i);
}

/*MD5 所要求的非线性运算子程序：同上*/
/*MD5 所要求的置换与循环子程序：同上*/
/*计算信息摘要的 MD5 程序：同上*/
/*--------------------主程序--------------------*/
main(){
    union long_char
        { unsigned long ceo;
          unsigned char cfo[4];
        }kof;
    unsigned long a1, b1, c1, d1;
    unsigned long cai, pai;
    void md5(void);
    unsigned long encode(unsigned long aa, unsigned long bb, unsigned long nn);
    md5();
    printf("Input your public key:");
    scanf("%ld, %ld", &cai, &pai);
    kof.cfo[0]=encode(hai[0], cai, pai);
    kof.cfo[1]=encode(hai[1], cai, pai);
    kof.cfo[2]=encode(hai[2], cai, pai);
    kof.cfo[3]=encode(hai[3], cai, pai);
    a1=kof.ceo;
    kof.cfo[0]=encode(hai[4], cai, pai);
    kof.cfo[1]=encode(hai[5], cai, pai);
    kof.cfo[2]=encode(hai[6], cai, pai);
```

```
    kof.cfo[3]=encode(hai[7], cai, pai);
    b1=kof.ceo;
    kof.cfo[0]=encode(hai[8], cai, pai);
    kof.cfo[1]=encode(hai[9], cai, pai);
    kof.cfo[2]=encode(hai[10], cai, pai);
    kof.cfo[3]=encode(hai[11], cai, pai);
    c1=kof.ceo;
    kof.cfo[0]=encode(hai[12], cai, pai);
    kof.cfo[1]=encode(hai[13], cai, pai);
    kof.cfo[2]=encode(hai[14], cai, pai);
    kof.cfo[3]=encode(hai[15], cai, pai);
    d1=kof.ceo;
    printf("The Signature after decryption is:\n");
    printf("%lx, %lx, %lx, %lx\n", a1, b1, c1, d1);
    if (a1==aa&&b1==bb&&c1==cc&&d1==dd)
      printf("\nThis test is validated by signature !");
    getch();
}
/*************************************************
```

(3) **生成一对不大于** 8 **位数的** RSA **非对称密钥的程序**。

随机选取不大于 4001 **的小素数** p **和** q，**计算** n=pq **和** fi=(p−1)(q−1)，**然后随机选取与** fi **互素的公钥** e **并算出相应的私钥** d，**以供上面程序演示之用**。

```
*************************************************/
#include "stdio.h"
#include "time.h"
#include "stdlib.h"
#include "math.h"
static int r1, r2;
/*小素数表*/
const static int PrimeTable[523]=
    {                                          109,  113,  127,
        131,  137,  139,  149,  151,  157,  163,  167,  173,  179,
        181,  191,  193,  197,  199,  211,  223,  227,  229,  233,
        239,  241,  251,  257,  263,  269,  271,  277,  281,  283,
        293,  307,  311,  313,  317,  331,  337,  347,  349,  353,
        359,  367,  373,  379,  383,  389,  397,  401,  409,  419,
        421,  431,  433,  439,  443,  449,  457,  461,  463,  467,
        479,  487,  491,  499,  503,  509,  521,  523,  541,  547,
        557,  563,  569,  571,  577,  587,  593,  599,  601,  607,
        613,  617,  619,  631,  641,  643,  647,  653,  659,  661,
        673,  677,  683,  691,  701,  709,  719,  727,  733,  739,
        743,  751,  757,  761,  769,  773,  787,  797,  809,  811,
        821,  823,  827,  829,  839,  853,  857,  859,  863,  877,
```

```
 881,  883,  887,  907,  911,  919,  929,  937,  941,  947,
 953,  967,  971,  977,  983,  991,  997, 1009, 1013, 1019,
1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087,
1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153,
1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229,
1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297,
1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381,
1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453,
1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523,
1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597,
1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663,
1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741,
1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823,
1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901,
1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993,
1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063,
2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131,
2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221,
2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293,
2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371,
2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437,
2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539,
2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621,
2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689,
2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749,
2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833,
2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909,
2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001,
3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083,
3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187,
3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259,
3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343,
3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433,
3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517,
3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581,
3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659,
3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733,
3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823,
3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911,
3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001
    };
main()
  {
```

```
    unsigned long p, q, n, fi, d, e;
    unsigned long pq(void);
    unsigned long ibm();
    unsigned long euclid(unsigned long ee, unsigned long ff);
    q=pq();
    p=pq();
    while(q==p) p=pq();
    printf("p=%ld, q=%ld\n", q, p);
    n=p*q;
    fi=(p-1)*(q-1);
    printf("n=%ld, phi=%ld\n", n, fi);
    e=ibm();
    while(e>=fi||e%p==0||e%q==0)
    e=ibm();
    d=euclid(e, fi);
    while(d==0)
      {e=ibm();d=euclid(e, fi);}
     if(d<0)
       d=d+fi;
     printf("d=%ld, e=%ld}\n", d, e);
    getch();
}

/*从小素数表中随机挑选 p 和 q 的子程序*/
unsigned long pq(void)
  { int z, nn;
    char (*p)[24];
    time_t curtime;
    int cia();
    time(&curtime);
    p=ctime(&curtime);
    r1=(*p)[17]*100+(*p)[18];
    z=cia();
    nn=PrimeTable[z];
    return(nn);
}

/*挑选 p 和 q 时使用的随机抽样器*/
int cia(void)
  { r1=(3*r1)%523;
    return(r1);
}
```

```
/* 用系统内置的 16 位随机数发生器产生密钥 e */
unsigned long ibm(void)
   { int i;
     randomize();
     r2=random(65536);
     return(r2);
}

/* 用欧基里德算法求 ee 的模 ff 逆元 dd 的子程序 */
unsigned long euclid(unsigned long ee, unsigned long ff)
   { unsigned long x2, x3;
     unsigned long y2, y3;
     unsigned long t2, t3;
     unsigned long qq;
     int k=0;
     x2=0, x3=ff;
     y2=1, y3=ee;
er:while(1)
   {
     if(y3==0)
     {return(0);}
     if((y3==1)&&(k%2==1))
        return(y2+ff);
     if((y3==1)&&(k%2)==0)
        return(y2);
     qq=floor(x3/y3);
     t2=x2-qq*y2;
     t3=x3-qq*y3;
     x2=y2, x3=y3;
     y2=t2, y3=t3;
     k++;
   }
}
```

B.6　Shamir 秘密门限共享方案设计

1. 共享系统的构建

随机生成一个 10 **位大素数** p **和四个** 8 **位的随机数** a_4、a_3、a_2、a_1；**需要隐藏的秘密数是** a_0。**以它们为系数构成一个** 4 **次方程式**：

$$f[x\_]:=\text{Mod}[a4*x\wedge 4+a3*x\wedge 3+a2*x\wedge 2+a1*x+a0,\ p]$$

求出曲线上的任意 8 **个点**：(x_1, y_1)，(x_2, y_2)，…，(x_8, y_8)。**例如**：

```
x1= Random[Integer, {10∧5, 10∧6}]; …
y1=f(x1); …
```

各个 x_i 是各用户的公钥，而各个 y_i 则是该用户的私钥，或者叫做该用户所分得秘密的份额。

2. 秘密如何被提取

方法 1　求解 5 阶矩阵方程(即 5 元联立方程组)：

```
Solve[{{{1, x1, x1∧2, x1∧3, x1∧4}, {1, x2, x2∧2, x2∧3, x2∧4},
    {1, x3, x3∧2, x3∧3, x3∧4}, {1, x4, x4∧2, x4∧3, x4∧4},
    {1, x5, x5∧2, x5∧3, x5∧4}}. {{s0}, {s1}, {s2}, (s3), {s4}}
    ==={{y1}, {y2}, {y3}, {y4}, {y5}}, Modulus
    == p}, Mode -> Modular]
```

其解为{s0, s1, s2, s3, s4}, s0，即为所要提取的秘密。

方法 2　采用拉格朗日插值多项式(见式(6-20)～(6-22))：

```
g[x_]:=InterpolatingPolynomial[{{x1, y1}, {x2, y2}, {x3, y3},
        {x4, y4}, {x5, y5}}, x]
```

将 0 代入，得到的常数项即为所求之秘密：

```
%/.x->0
```

但以分数形式给出，还需要作模 p 计算：

```
a0=Mod[Numerator[%] * PowerMod[Denominator[%], -1, p], p]
```

3. 验证

如果少于 5 个用户到场，方程数少于未知数个数，就得不到唯一解；如果等于或多于 5 个用户到场，不难验证，不论用哪 5 个用户的数据，算出的结果都是一样的。

附录 C　习题参考答案

习题 1

1. “HPHTWWXPPELEXTOYTRSE”
2. “VPXZGIAXGVWPUBTTMJPWGZITWZT”
3. “ZLBT”
4. $K=\begin{pmatrix}3 & 4 & 6\\21 & 15 & 14\\20 & 23 & 5\end{pmatrix}$
5. “meet me on the sabbath we will discuss the plan”

习题 2

1. **两种：分组(块)加密、序列加密。**

2. **指密钥序列是无限长的真正的随机序列，密钥使每位密文都变得无规律可循。对于这样的系统，破译者完全无法从已有的密文中，或明、密文对照中找到规律，不能为未知的密文破译提供任何有价值的信息。**

3. **序列的复杂度指序列有足够繁多的变化花样。算法的复杂度指一种算法的计算工作量随序列长度 n 以何种方式增大，当计算量随序列长度 n 以高于多项式的方式增大时，则认为该算法复杂度高，属于 Np 难解问题，从而在理论上保障了系统是难以破译的。**

4. **全 0 的序列。**

5. $2^{48}-1$

6. **对于初值 0000，0001，0010，0011 只能产生全 0 序列，其他初值产生周期为 3 的序列。**

7. $C_0=1$，$C_1=0$，$C_2=1$，**初值为**“111”。

习题 3

1. (1) $C_0=$11110000 11001100 10101010 0000；
   $D_0=$10101010 11001100 11110000 0000
   (2) $C_1=$1110000 11001100 10101010 00001；
   $D_0=$0101010 11001100 11110000 00001
   (3) $K_1=$00001011 00000010 01100111 10011011 01001001 10100101

2. S_3 **输出** 1100；S_7 **输出** 1111。

3. $Y_1^{(1)}=X_1^{(2)}=10011110\ 10010111$，$Y_2^{(1)}=X_3^{(2)}=01111000\ 01000100$
$Y_3^{(1)}=X_2^{(2)}=01010010\ 10100101$，$Y_4^{(1)}=X_4^{(2)}=00101101\ 01001010$

4.（略）

5.（B5 70 78 40 7A 51 B2 37 7E 31 47 DF 90 06 82 84）$_{16}$

习题 4

1.（1）$n=33$；$\phi(n)=20$；$d=3$；$C=14$
（2）$n=55$；$\phi(n)=40$；$d=27$；$C=14$
（3）$n=77$；$\phi(n)=60$；$d=53$；$C=57$
（4）$n=143$；$\phi(n)=120$；$d=11$；$C=106$

2. F 的编码为 6，$C=41$；接收方：$41^{77} \bmod 119=6$，译码即原来字符 F。

3. 5

4. 由 $\Phi(n)=(p-1)(q-1)=pq-(p+q)+1$ 得 $(p+q)=n-\Phi(n)+1$，而 $pq=n$，因此知 p 与 q 是二次方程 $x^2+[n-\Phi(n)+1]x+n=0$ 的两个根。代入数据解方程得 $p=33\ 073$，$q=77\ 069$。

5. 提示：因 $e\cdot d=k\Phi(n)+1$，可令 $e\cdot d-1=k\Phi(n)=2^r\cdot m$，适当选择 r 总能使 $m=[k\Phi(n)]/2^r$ 为奇数。任意选择与 n 互素的数 a，应满足欧拉定理：

$$a^{k\Phi(n)}=a^{2^r m}=1 \bmod n$$

一般 $a^m\neq 1 \bmod n$，从 a^m，a^{2m}，$a^{4m}\cdots$ 一直计算到 $a^{2^r m}$，这个队列在 $\bmod n$ 运算下总会在某一处由不等于 1 变为等于 1。设这个地方是在 a^{tm} 处，即：

$$a^{tm}\neq 1 \bmod n$$

而

$$a^{2tm}=1 \bmod n$$

（实际上该队列在 a^{tm} 前面的模幂值都不等于 1，而后面都等于 1）。

令 $a^{tm}=x$，则 $a^{2tm}=x^2=1 \bmod n$ 成为一个二次剩余问题。由

$$x^2-1=(x+1)(x-1)=0 \bmod n$$

知 n 的两个因子分别存在于 $(x+1)$ 与 $(x-1)$ 之中，通过计算最大公约数就可求出 $p=\gcd\{(x+1),n\}$，代入数据求得 $p=37$ 而 $q=n/p=43$。

6.（1）利用欧拉定理，算得密文：

$$C=M^N \bmod N=M^{N\pm k\Phi(N)} \bmod N=M^{N \bmod \Phi(N)} \bmod N \tag{1}$$

由 $P\cdot P'=1 \bmod (Q-1)$ 知，若 $M^P \bmod Q=C_1$，则 $C_1^{P'}=M \bmod Q$；由 $Q\cdot Q'=1 \bmod (P-1)$ 知，若 $M^Q \bmod P=C_2$，则 $C_2^{Q'}=M \bmod P$。

于是，利用费尔玛定理 $M^{Q-1}=1 \bmod Q$，就有：

$$\begin{aligned}C_1&=M^P \bmod Q=M^P\cdot M^{Q-1} \bmod Q=M^{P+Q-1} \bmod Q\\&=M^{N-\Phi(N)} \bmod Q=M^{N \bmod \Phi(N)} \bmod Q\end{aligned} \tag{2}$$

比较式(1)与式(2)可知 $C_1=C \bmod Q$，所以

$$M=C_1^{P'} \bmod Q=C^{P'} \bmod Q$$

同理有

$$M=C_2^{Q'} \bmod P=C^{Q'} \bmod P$$

(2)、(3)(略)

7. $2^{10}=1024\equiv 1 \bmod 341$，$2^{340}=(2^{10})^{34}\equiv 1 \bmod 341$

所以 $2^{341}\equiv 2 \bmod 341$

而 $341=31\times 11$，故必要条件不成立。

8. (1) (50，31)　　(2) 29

9. 6689

10. 无穷远点$\varnothing$

11. $y=((8,3),(10,2))$

12. $P_1+P_2=\left(-\frac{144}{25},-\frac{504}{125}\right)$　　$2P_1=\left(\frac{25}{4},-\frac{35}{8}\right)$

13. 加密：$C=(696,1856,1712,1395,500,971)\cdot(101010)^T \bmod 2000=908$

解密：因为 $908\times 69=652 \bmod 2000$，而 $652=500+128+24$

所以 $M=(101010)$。

习题 5

1. 签名结果为$(r,s)=(1490,1777)$。

2. (1) MD4 使用三轮运算；MD5 使用四轮运算。

(2) MD4 第一轮没有加法常量，第二轮运算中加法常量相同，第三轮运算中每步迭代使用的加法常量相同(尽管不同于第二轮)；MD5 的四轮 64 步迭代中均使用不同的加法常量。

(3) MD5 使用四个基本逻辑函数；MD4 使用三个基本函数。

(4) MD5 中每步迭代结果都与前一步结果相加；MD4 中没有。

3. 签名为(94，97)。

4. 提示：首先 k 相同则对于两个消息 m_1 和 m_2 签名的 r 相同。由 $(s_1-s_2)k=m_1-m_2$，分析同余式 $\gcd(s_1-s_2,p-1)$(其结果通常很小)，根据 r 分析计算出 k 值，再由 s 分析得到 x 值。

5. 先签名后加密。

6. 对消息摘要进行数字签名比直接签名更高效。

7. (略)

8. $1-6\sqrt{-24\ln 0.95}$

9. 提示：

(1) 攻击目的是寻找一对密钥(K_i,K_j)，使得 $E_L(E_K(m_0))=E_L(c_0)=E_{K_0}(m_0)$，从而获取 K_0。因已知(m_0,c_0)，对所有可能密钥 K 都计算 $E_{K_i}(m_0)$ 和 $D_{K_j}(c_0)$ (K_i 和 K_j 是密钥集中任一密钥)，若两者相同，那么由题意，易找到 K_0。

(2) 对可能密钥 K 分别计算 $E_K(m)$和 $E_K(E_K(m))$，两者比较，相同者，$E_K(E_K())$即解密函数。

10. 99.3%，3723。

习题 6

1. $b_U=50$，$b_V=44$，$k_{UV}=75$

2. $K_{AB}=14$，$K_{AC}=0$，$K_{BC}=6$

3. $k=13$

4.（1）C 截获（$E_{PB}(m)$，A），改为（$E_{PB}(m)$，C）发送给 B，将会收到 B 回复的（$E_{PC}(m)$，B）。于是，C 可以利用私钥 S_C 解密得到 m。

（2）改变通信格式后，增强了保密性，增加了认证和完整性校验的功能。

5. $k_1=2$，$k_2=8$，$k_3=9$

习题 7

1.（略）

2.（略）

3. 设重复的密钥序列为 K，A 和 B 为明文，X 和 Y 是相应的密文，即 $K \oplus A=X$，$K \oplus B=Y$，则 $A=X \oplus Y \oplus B$。当攻击者截获到两个密文后，如果知道 A 的某些部分，可恢复出 B 的对应部分。

4. ISAKMP 本身没有指定特定的密钥交换算法，而是包含了一些模式，可以使用不同的密钥交换算法。Oakley 是最初的 ISAKMP 中规定的密钥交换算法。

5.（1）使用常规的对称加密算法。

（2）通过使用 nonce 值。

（3）使用公钥认证。

（4）加密数据。

（5）攻击者必须同时伪造密钥及其对应的 IP 地址。

6. CFB 模式避免了填充位添加和去掉的过程。

习题 A

1. $(1465)_7$

2. 191×109

3. 因为 $a\equiv b(\bmod m)$，$c\equiv d(\bmod m)$

所以有

$$s,t\in Z,\ a=b+sm,\ c=d+tm$$

所以

$$a\pm c=b\pm d+(s\pm t)m$$

所以

$$a\pm c\equiv b\pm d \bmod m$$

4. 36

5. 1，8，55，62

6. 82 mod 121

7.（略）

8. 26，47，68，89

9. 27

参 考 文 献

[1] C. E. Shannon. Communication Theotry of Secrecy System[J], Bell Syst Tech J, 28. 1949: 656 - 715.

[2] **卢开澄. 计算机密码学**. 2 **版**. [M]. **北京: 清华大学出版社**, 1998.

[3] Bruce Schneier. **应用密码学协议——算法与** C **源程序**[M]. **吴世忠, 祝世雄, 张文政, 等译. 北京: 机械工业出版社**, 2005.

[4] **章照止. 现代密码学基础**[M]. **北京: 北京邮电大学出版社**, 2004.

[5] Wade Trappe, Lawrence C. Washington. **密码学概论**[M]. **邹红霞, 等译. 北京: 人民邮电出版社**, 2004.

[6] **吴伟陵. 信息处理与编码**[M]. **北京: 人民邮电出版社**, 1999.

[7] **王育民. 保密学——基础与应用**[M]. **西安: 西安电子科技大学出版社**, 1990.

[8] **杨义先. 现代密码新理论**[M]. **北京: 科学出版社**, 2002.

[9] R. C. Merkle. Secure Communication over Insecure Channels[J]. Communications of the ACM, v. 21(4), 1978: 294-299.

[10] W. Diffe and M. E. Hellman. New Directions in Cryptography[J]. IEEE Transactions on Information Theory, v. IF-22(. 6), 1976: 644 - 654.

[11] R. C. Merkle and M. E. Hellman. Hiding Information and Signatures in Trapdoor Knapsacks[J]. IEEE Transactions on Information Theory, v. 24(5). 1981: 465 - 467.

[12] **王新梅, 肖国镇. 纠错码——原理与方法**[M]. 2 **版. 西安: 西安电子科技大学出版社**, 2001.

[13] **肖攸安. 椭圆曲线密码体系研究. 武汉: 华中电子科技大学出版社**, 2006.

[14] A. Menezes, T. Okamoto, S. Vanstone. Reducing elliptic curves logarithm to a finite fields[J]. IEEE Transactions on Information Theory, 39. 1993: 1639 - 1646.

[15] P. Gaudry, F. Hess, N. Smart. Constructive and destructive facets of Weil descent on elliptic curves [J]. Journal of Cryptology, 15(1). 2002: 19 - 45.

[16] S. Erkay, A. Thomas. Generating elliptic curves of prime order [C]. Berlin: Springer-Verlag 2001.

[17] R Schoof. Eliptic Curves over Finite Fields and Computation of Square Roots mod p [J]. Mathematics of Computation. 44. 1985: 483 - 494.

[18] R Schoof. Counting Points on ElipticCurves over Finite Fields [J]. Joumal of Theorie des Nombres de Bordeaux. 1995. (7): 219 - 254.

[19] L. Dewaghe. Remarks on the Schoof-Elkies-Atkin Algorithm[J]. Mathematics of Computation, 67, 1998: 1247 - 1252.

[20] William Stallings. **密码编码学与网络安全——原理与实践**. 3 **版. 刘玉珍, 等译. 北京: 电子工业出版社**, 2005.

[21] 阎喜戎. 信息安全原理及应用[M]. 北京：清华大学出版社，2003.
[22] 杨波. 现代密码学[M]. 北京：清华大学出版社，2003.
[23] H. Krawczyk，M. Bellare，R. Canetti. HMAC：Keyed-Hashing for Message Authentication. Network Working Group. RFC 2104，1997.
[24] ITU-T Recommendation X. 509. Information Technology - Open Systems Interconection-The Directory：Authentication Framework，1997.
[25] Carlisle Adams，Steve Lloyd. 公开密钥基础设施——概念标准和实施[M]. 冯登国，译. 北京：人民邮电出版社，2001.
[26] 关振胜. 公钥基础设施 PKI 与认证机构 CA [M]. 北京：电子工业出版社，2002.
[27] C. Adams，S. Farrell，T. Kause，T. Mononen. Internet X. 509 Public Key Infrastructure Certificate Management Protocol (CMP). Network Working Group，September 2005.
[28] C. Adams，S. Farrell. Internet X. 509 Public Key Infrastructure，Certificate Management Protocols. Network Working Group，March 1999.
[29] ISO/TC68/SC2. Banking-Certificate Management Part 1：Public Key Certificates. ISO/DIS 15782-1，1999.
[30] ITU-T，Draft Revised Recommendation X. 509. Draft Revised ITU-T Recommendation X. 509 ISO/IEC 9594-8：Information Technology-Open System Interconnection - The Directory：Public-Key and Attribute Certificate Frameworks. Version 4，2000.
[31] Andrew Nash. 公钥基础设施(PKI)实现和管理电子安全[M]. 张玉清，等译. 北京：清华大学出版社，2002.
[32] Messaoud Benantar. 互联网公钥基础设施概论[M]. 张千里，译. 北京：人民邮电出版社，2003.
[33] 3GPP TS 33. 105 version 700：Cryptographic Algorithm Requirements.
[34] 3GPP TS 35. 201 version 700：Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 1：f8 and f9 Specification.
[35] 3GPP TS 35. 202 version700：Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 2：KASUMI Specification.
[36] 毛光灿，何大可. 3G 核心加密算法 KASUMI 算法[J]. 通信技术，11(131). 2002：92 - 97.
[37] 3GPP TS 35. 203 version4. 0. 0：Report on the Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms.
[38] 徐明. 网络信息安全[M]. 西安：西安电子科技大学出版社，2006.
[39] Naganand Doraswamy，Dan Harkins. IPSec：the new security standard for Internet. intranet and virtual private networks，1999.
[40] S. Kent，R. Atkinson. Security Architecture for the Internet Protocol. RFC2401，1998.
[41] S. Kent，R. Atkinson. IP Authentication Header. RFC2402，1998.
[42] S. Kent，R. Atkinson. IP Encapsulating Security Payload. RFC2406，1998.

[43] D. Harkins, D. Carrel. The Internet Key Exchange (IKE). RFC2409, 1998.

[44] T. Dierks, C. Allen. The TLS Protocol Network Working Group. RFC2246, 1999.

[45] **山风，山月. 实作** ActiveX **组件** for ASP[M]. **北京：中国铁道出版社**，2001：181－188

[46] B. Ramsdell. S/MIME Version 3 Message Specification. RFC2633, 1999.

[47] **丁秀源，薛昭雄. 密码学与数论基础. 济南：山东科学技术出版社**，1993.

[48] Douglas R. Stinson. Cryptography Theory and Practic—Second Edition. **冯登国，译. 北京：电子工业出版社**，2003.

[49] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications. IEEE Computer Society, 2004.

[50] **孙淑玲. 应用密码学**[M]. **北京：清华大学出版社**，2004.